Frédéric Vallotti
Benjamin Bounous
N4

Projet Technique N4 - AI²R
Coupe de France de Robotique 2004
"Coconut Rugby"

Partie II :
Traitement d'image avec Caméra

Introduction
I LE PROJET
II LA CMUCAM 4
II-1 Description du module CMUcam 4
II-2 Pourquoi la CMUcam? 4
II-3 La CMUcam en résumé 4
II-3.1 Caractéristiques et conditions d'utilisation 4
II-3.2 Mécanisme Hardware 5
II-3.3 Mécanisme Software 5
II-3.4 La plaquette CMUCAM 6
II-3.5 Principe de fonctionnement 6
II-3.6 Fonctions les plus importantes utilisées 6
II-3.7. Les plus et les moins de cette caméra 7
III LA COMMUNICATION SERIE ASYNCHRONE 7
III-1 Comment ça marche 7
III-2 Port Série Asynchrone SCI ST7 9
IV-COMMUNICATION CAMERA-ST7 12
IV-1 Initialisation de la caméra 12
IV-2 Initialisation du st7 13
V PROGRAMATION ST7: PRINCIPES GENERAUX 14
VI CAMERA MODE DE FONCTIONNEMENT 16
VI-1 Fonction conversion 16
VI-2 Fonction send 18
VI-3 Fonctions receive 19
VI-4 MODE RM 3 18
VI-5 MODE RM 7 19
VI-6 MODE Poll Mode PM 19
VI-7 Mode paquet de données MM 19
VII Développement de la première stratégie 21
VII-1 Fonction traquer une couleur 21
VII-2 Où sont les balons ? 21
VII-3 Explication du contenu de la fonction ramnène 22
VII-3.1 La fonction triballe 23
VII-3.2 calcul d'angle et de distance 30
VII-3.3 La fonction alerte 34
VIII. Description des programmes secondaires: DEUXIEME STRATEGIE 35
VIII-1 La fonction ia 35
VIII-2 La fonction calibrage 38
VIII-3 Calcul de la positon du robot 42
VIII-4 La fonction fire 45
VIII-5 La fonction 2coups 48
CONCLUSION 51
1. Frédéric Vallotti 51
2. Benjamin Bounous 52
ANNEXE 1 : fonction ramene corrigée……54
ANNEXE 2 : fichier math.c…...……74
ANNEXE 3 : CAMERA……78
ANNEXE 4 : @ utiles……..90

INTRODUCTION

Dans le cadre du projet E=M6, doter le robot d'une caméra CMUCAM n'était pas un nouveau projet en soi car elle fut également implantée lors du concours de l'année dernière. Nous avons eu la chance de pouvoir récupérer le travail fourni par les membres de l'année dernière sur cet appareil qui, pour nous, était outil tout nouveau.

Nous avons choisi de reprendre la caméra cette année encore car c'est l'un des moyens les plus sûrs que nous avons trouvé pour remplir la lourde tâche de détection de balles. En effet, la caméra reste l'un des moyens les plus simples pour permettre au robot de voir et d 'analyser l'environnement qui l'entoure et de pouvoir ainsi « réfléchir » quant à la trajectoire qu'il doit emprunté pour remplir sa mission.

Ce rapport a pour objectif de vous dresser toutes nos réflexions et nos stratégies de la façon la plus explicite afin que ceux qui s'intéresseront à ce projet s'en imprègnent et aillent de l'avant.

Vous trouverez donc ci-après, le détail de nos codes dûment commentés mais aussi les étapes qui nous ont conduit à l'écriture de ces codes. Ces derniers commencent d'abord par l'élaboration des primitives de premier niveau, en passant par la communication avec le ST7 puis enfin l'élaboration d'une partie de l'intelligence artificielle. Ce projet est avant tout un projet de programmation en C et en assembleur. Vous apprendrez accessoirement comment nous avons géré la dualité soft / hard avec le ST7 ainsi que la communication série avec la caméra déjà étudiée l'année dernière, et enfin nous vous donneront quelques compléments sur l'utilisation de la famille du ST7 que vous devez maintenant bien connaître.

Nous vous souhaitons une bonne lecture.

II LA CMUCAM

II-1 Description du module CMUcam

La caméra est composée de 2 parties visibles sur l'image ci-contre:

1. Un petit circuit imprimé (board) sur lequel est fixée la lentille.
2. Un circuit imprimé un peu plus grand sur lequel est connecté le premier board, et se trouve le microcontrôleur (SX28), le circuit d'interfaçage MAX232 vers le bus RS 232, 2 leds, un bouton ON/OFF et différents connecteurs non utilisés (cartes optionnelles, re-programmation du SX28, contrôle servomoteur…)

II-2 Pourquoi la CMUcam?

On sait que la caméra permet d'avoir une vue 2D du réel par l'intermédiaire d'une image constitué d'éléments élémentaires appelés pixels. Le challenge de l'année dernière a été de mettre en place des dispositifs externes telles une RAM et une EEPROM afin de stocker les images obtenues et ensuite d'analyser ces dernières pour avoir les informations adéquates sur les objets recherchés sur le terrain. Toutes ces étapes sont relatées dans le rapport 2002 du projet caméra. Un inconvénient de cette méthode est de devoir gérer les adressages mémoire nécessaires à la récupération de l'information et d'y synchroniser l'exécution des procédures.

Cette année nous avons opté pour un module ayant déjà des modules de stockage et de traitement de l'image intégré et il "a donc suffit" de concentrer nos efforts afin de contrôler le module pour aider le robot dans sa tâche.

Ce module est la CMUCAM qui présente les avantages que nous venons de citer et qui nous coûte environ 100 euros.

II-3 La CMUcam en résumé

II-3.1 Caractéristiques et conditions d'utilisation

Dimension du terrain	300x210cm
Champ de vision de la caméra	lentille 4,9mm F2.8
Caméra Posée sur Support Balise FIXE	Non. Posé sur le robot
Dimension de l'image	80x143

Type de données traitées	ASCII
Type de caméra	OV6620OmnivisionCMOS caméra
Résolution caméra	352*288
Type microcontrôleur	SX28 75Mhz 8bits 1bit d'arrêt
Taux max de rafraîchissement	17 frames/s
Mémoire du programme	2Ko EPROM Flash programmable
Mémoire de l'image	136octets de RAM
Langage de programmation	C
Module d'émission Balise - Robot	Aucune
Alimentation en tension du module	6V
Courant fourni au module	180mA
Interfaçage caméra microcontrôleur	Aucune
Quelles sont les registres de la caméra	cf. doc de OMNIVISION

Temps d'initialisation et d'ajustement de la caméra à chaque changement drastique lumineux de l'environnement: 5s

II-3.2 Mécanisme Hardware

Le bus de données, les ports synchrones et les bus de configuration relient directement la caméra au microcontrôleur ce qui permet à ce dernier de traiter les données en temps réel et de le mettre à disposition via le chipset MAX232 et les ports séries TTL.

II-3.3 Mécanisme Software

La CMUcam est un microcontrôleur interfacé avec une caméra CMOS permettant d'extraire des données à partir de la vidéo fournit par l'œil de la caméra. On utilise une caméra CMOS au lieu d'une CCD à cause de sa capacité à intégrer des circuits additionnels comme un détecteur ou des convertisseurs analogiques numériques.

La caméra dispose d'une zone de stockage de l'image d'une capacité de 101 376 pixels et supporte des résolutions supérieures à 352*288 avec une fréquence de rafraîchissement maximale de 60 frames par sec.

Comme notre caméra doit permettre de commander le robot, il est connecté à un processeur extérieur qui doit être configuré afin de spécifier les niveaux de RGB: LE SX28.

Le SX28 a une architecture Harvard (contrairement à Von Neumann) i.e. les données et les instructions sont dans des zones mémoires différentes. Ceci lui permet d'avoir un bus de données de 8bits et un bus d'instructions de 12 bits.

On peut remarquer que la zone de stockage de l'image dans le microcontrôleur (136octets=136pixels) est faible. De plus, seulement 80 octets sont réservés à l'image et 56 au traitement. Afin de traiter les données, le SX28 analyse tour à tour les lignes de l'image capturé en sélectionnant 20 paquets de pixel RGB.

II-3.4 La plaquette CMUCAM

Etant donné que la plaquette est munie d'un régulateur 5V, on alimentera entre 6 et 7 volts avec une alimentation pouvant fournir au moins 200mA. Mais à cause du Servo port qui ne dispose pas de ce régulateur, l'alimentattion se fera à 5-6V maxi.

Les ports TTL permettent la communication entre le microcontrôleur et un processeur externe. Ils peuvent être utilisés pour communiquer avec un microcontrôleur marchant en mode TTL mais dans ce cas il peut être nécessaire d'enlever le Max232.

Les données de la CMUcam peuvent être traitées simultanément (par exemple chercher une couleur précise et avoir la couleur moyenne sur un ensemble de données) par plusieurs plaquettes à condition de les mettre en disposition maître - esclave.

Lorsque le jumper 1 est positionné la caméra devient esclave. Seule la plaquette maître dispose des registres de la caméra.

Les jumpers 2 et 3 doivent être, si c'est possible, non positionnés afin d'avoir une meilleure résolution et un meilleur taux de capture de l'image.

Lorsque la caméra est utilisée en plein air, il est nécessaire d'utiliser un filtre IR pour diminuer le niveau de lumière ambiant.

La lentille attachée à la caméra détermine son champ de vision.

II-3.5 Principe de fonctionnement

La caméra, selon des paramètres programmés, analyse l'image et retourne des informations. Il est possible de lui faire analyser toute l'image ou seulement une partie de celle-ci. Les informations retournées sont, par exemple, la taille et position d'objets qui contiennent une gamme de couleur recherchée, la confiance sur cette mesure, la couleur moyenne et la déviation pour chaque composante Rouge/Verte/Bleue de la partie analysée.

Il n'est en aucun cas nécessaire de capturer une image complète en mémoire et d'analyser celle-ci. Il est toujours possible de capturer une image complète mais cette fonction n'est utile que pour de la mise au point ou des calibrages dimensionnels (c'est-à-dire savoir quel surface du terrain la caméra observe en fonction de son positionnement sur le robot).

Il existe un environnement en Java pour prendre en main le système et faire des essais. (voir annexe « adresses utiles »).

II-3.6 Fonctions les plus importantes utilisées

(Pour plus de description, voir « CMU-CAM – Guide d'utilisation »)

CR : Camera Register. règle les registres internes de la caméra.

DF : Dump Frame. Pour capturer une image complète. Pour le debug ou le calibrage dimensionnel.
GM : Get Mean. Renvoie la valeur moyenne et la déviation de chaque composante Rouge/Verte/Bleue.
L1 1 ou L1 0 : allume ou éteint la LED rouge. Cette commande est utile pour le debug (vérification de la communication série)
SW : Set Window : Définit la zone rectangle où l'analyse de l'image se fera.
TC : Track Color. Renvoie les coordonnées d'un rectangle où se situe la couleur définit par la commande CR, ainsi qu'une valeur de confiance.

II-3.7. Les plus et les moins de cette caméra

Les plus :

- Intègre suffisamment de fonctions pour normalement ne pas à avoir besoin de code pour faire de l'analyse d'image.
- Facile à programmer
- Evolutive grâce aux options (servomoteur)
- Dialogue par simple bus série RS232, vitesse programmable jusqu'à 115000 bauds
- Prix

Les moins :

- La résolution peut être limite pour analyser des objets distants.
- Dynamique en contraste/luminosité pas très bonne.
- Sensibilité à la source d'éclairage.
- Fonction Auto Gain / White Balance pas toujours très efficace.
- Bus série dédié. La caméra ne gère pas d'adressage. On ne peut donc pas brancher d'autres appareils utilisant un bus série sur le même bus que la caméra.
- Si l'on veut charger une image complète, prends 5 à 6 secondes.
- Angle de vue faible (environ 45°)

III LA COMMUNICATION SERIE ASYNCHRONE

III-1 Comment ça marche

La communication série asynchrone est utilisée pour communiquer avec un ordinateur distant ou local. Dans la cas des microcontrôleurs, le système de communication série est le SCI(Serial Communication Interface) que nous décriront dans la partie consacrée au microcontrôleur. Il permet la transmission de données numériques en série, c'est-à-dire les bits transmis les uns après les autres, sur une paire de fils, le bit 0 étant transmis en premier.

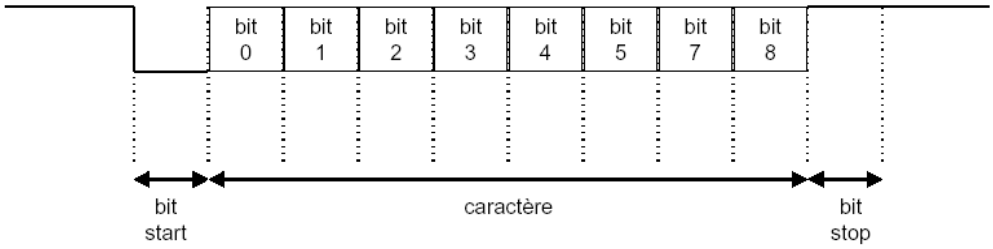
Pour échanger les données dans les deux directions simultanément, il faut utiliser 3 fils (un dans chaque direction et un fil de masse). Ce sera ce cas qui nous intéressera dans la mesure où notre microcontrôleur devra interroger celui de la caméra et que ce dernier devra répondre en envoyant l'information demandée.

N.B : il est très important de brancher le fil de masse avec la masse de la plaquette du ST7. cela peut paraître trivial, mais si la caméra et le ST7 n'ont pas de masse commune, les deux entités ne communiquent pas.

Le caractère asynchrone de la transmission vient du fait que les données sont subdivisées en caractères de 7,8 ou 9 bits et que chaque caractère est transmis indépendamment avec sa propre référence temporelle.

La synchronisation des bits à la réception est assurée à l'aide d'un bit initial connu: le bit de départ (**Start Bit**). C'est à partir de la position du bit de départ que le récepteur se synchronise et échantillonne les autres bits du caractère. Pour valider la réception, chaque caractère est terminé par un bit final lui aussi connu: le bit d'arrêt (**Stop bit**).

Voici le schéma d'un caractère transmis:



Au repos la ligne est positionnée à 1. Le passage de la ligne au niveau bas indique le démarrage de la transmission. Ceci correspond au bit de start. Une fois la donnée envoyée, la ligne passe au niveau 1 ce qui correspond au bit de stop.

Une interface série est composée de:

- Un module qui gère la vitesse de transmission

- Un module émetteur avec un registre de décalage d'émission pour la conversion parallèle/série(un octet en série de bits)
- Un module récepteur avec un registre de réception pour la conversion série/parallèle (une série de bits en un octet)

Comme pour le st7, ces deux registres peuvent être physiquement identiques et ne sont pas accessibles au programme même en assembleur.

III-2 Port Série Asynchrone SCI ST7

D'une façon très générale, voici les étapes à suivre pour l'utilisation d'une interface série:

1- Pour pouvoir envoyer et recevoir des données il faut :

-Activer le récepteur

-Activer l'émetteur

-Fixer la vitesse de transmission

-Choisir la longueur des données à échanger : une longueur de 8 bits ou de 9 bits. La longueur la plus utilisée est celle d'un octet i.e. 8 bits. Généralement, cette initialisation se fait une seule fois par le programme.

2- Pour envoyer un octet, il suffit de l'écrire dans un registre spécial. Le registre d'émission.

3- L'arrivée d'un octet sur le port série est signalée par un bit d'état qui sera mis à 1 par hardware. L'octet reçu est alors disponible dans un registre spécial. Le registre de réception. Pour le ST7, le registre d'émission est aussi le registre de réception. Il est donc primordial de réceptionner chaque octet à la vitesse à laquelle ils sont envoyés par la caméra, et les stocker dans une variable quelconque.

4- Si une erreur se produit lors de la communication alors elle sera signalée par un bit d'état qui sera mis à 1 par hardware. Les deux causes d'erreur sont :

- *Le débordement* : des données arrivent alors que la donnée précédente n'a pas été lue par le programme.
- *Le format :* le récepteur indique une erreur de format si le bit de stop n'est pas détecté.
- Le récepteur reçoit un niveau 0 alors qu'il attend le bit de stop (niveau 1).
- Une erreur dans la vitesse de communication se traduit souvent par une erreur dans le format de la donnée reçue.

Concrètement vous trouverez dans l'annexe de ce document les extraits de Datasheet qui décrivent le fonctionnement et le mode d'emploi du SCI. Les descriptions ressemblent quasiment à ceux donnés ci-dessus, le SCI étant une norme de communication.

En résumé, Les registres permettant la communication sont:

Le registre de donnée SCIDR: une écriture dans ce registre transmet un mot et une réception se fait par la lecture du registre.

DR7	DR6	DR5	DR4	DR3	DR2	DR1	DR0

Le registre de contrôle SCICR1: permet entre autre de spécifier la longueur du mot à envoyer.

R8	T8	-	M	WAKE	-	-	-

Le registre de contrôle SCICR2: permet de rendre actif les routine de transmission et de réception.

TIE	TCIE	RIE	ILIE	TE	RE	SWU	SBK

Le registre d'état SCISR: accessible en lecture uniquement et décrit l'état de la communication.

TDRE	TC	RDRF	IDLE	OR	NF	FE	-

Le "Baud Rate Register" SCIBRR: permet de spécifier la vitesse de transmission. On peut aussi utiliser le SCIETPR et le SCIERPR. Le calcul des vitesses de transmissions est spécifié à la page 91 de la Datasheet du ST7.

SCP1	SCP0	SCT2	SCT1	SCT0	SCR2	SCR1	SCR0

Pour transmettre un mot: cf. Datasheet p.88
Pour recevoir un mot: cf. Datasheet p.89

Ce qu'il faut savoir :
Lorsque vous envoyez plusieurs octets, faites attention à les espacer d'une petite temporisation afin d'éviter les collisions.

Pour plus de renseignements nous vous conseillons de regarder la fonction d'initialisation du ST7 qui vous montre de quelle manière nous avons initialisé les registres, sachant que par la suite, et nous vous expliqueront pourquoi, nous avons élaboré 2 fonctions d'initialisation.

IV COMMUNICATION CAMERA-ST7

Comme nous l'avons vu précédemment, le st7 sera l'outil indispensable pour que le robot puisse communiquer avec la caméra en toute liberté et indépendamment du reste. Pour cela, il vous faudra d'abord bien vous familiariser avec les fonctions de la caméra et comprendre son mode de fonctionnement. (cf datasheet de la caméra en annexe). En effet, la caméra fonctionne sur le principe de reconnaissance des couleurs RVB. C'est-à-dire que l'image renvoyée est une matrice de pixels dont les octets contiennent les informations des composantes rouge, vert et bleu de l'image. Votre seul moyen de trouver un objet sera donc basé sur la couleur de ce-dernier.

Cependant, la caméra travail en langage ASCII. Toutes les données que vous lui enverrez devront donc être traduit caractère ASCII. Par exemple, si vous vous voulez qu'elle vous renvoie la moyenne des composantes RVB de l'image, vous devez lui envoyer la commande « GM ». pour cela, vous devez décomposer cette chaîne de caractères en lui envoyant « G », puis « M ». pour que le ST7 et la caméra fonctionnent correctement, il est impératif que ce protocole soit établi dans un sens comme dans l'autre. Les fonctions de ce niveau vous sont expliquées plus bas.

Pour que la caméra reçoive et envoie les octets correctement, il faut évidemment que le ST7 et la caméra aient les mêmes normes de communication i.e. le même protocole à savoir:

- Définir la longueur du mot à envoyer
- Définir la vitesse de transmission et de réception exprimée en bauds(bits/s)

Dans le cadre de la caméra on envoie des octets à une vitesse de 19200bauds. Ce choix est arbitraire et est le résultat d'un dilemme: La vitesse maximale de 115 000 bauds entraînait des pertes d'informations pour la caméra toutefois d'après la Datasheet du St7, une vitesse élevée assurait une meilleure transmission de la donnée.

Du point de vue Hardware, on utilise les ports TTL de la caméra et on règle les jumpers de façon à choisir la bonne vitesse de transmission. Sur le ST7, on utilise les ports SCI (ici le RDI et le TDO : ports PE0 et PE1). L'initialisation de la vitesse se fait à software.

Voici les deux programmes permettant d'initialiser nos deux appareils.

IV – 1 Initialisation de la caméra

La fonction d'initialisation de la caméra permet de rendre cette-dernière opérationnelle et prête à recevoir et transmettre des informations :

```
void initcam(void)
{
	tortue = depart_tortue;	//  initialisation de la fonction odométrie
	position_y = depart_position_y; //  initialisation de la fonction odométrie
	tempo(Temp);
	send(RS);	// reset caméra
	tempo(60*Temp);
	send (alive);
	tempo(60*Temp);
	send(SW); //  définit le tout le champ de vision de la caméra.
	tempo(60*Temp);
	send(alive);
	tempo(60*Temp);
	send(PM1);	//poll mode on : 1 seule trame renvoyée.
	tempo(60*Temp);
}
```

IV – 2 Initialisation ST7

Tout au long du projet nous avons eu des problèmes quant à l'initialisation du ST7. nous nous sommes rendus compte que le st7 et sa plaquette de simulation ne travaillait pas à la même vitesse. Nous avons donc élaboré deux fonctions selon que nous travaillions sur plaquette ou sur le vrai ST7.

Ceci est la fonction qui permet de faire fonctionner le ST7 seul sur le robot.

```
void initst7_seul(void)
{
	//PEDDR = 0x00; initialisé dans hctl.c
	//PEOR = 0x00;

MISCR1 = 0x10;	//IS11|IS10|MCO|IS21|IS20|CP1|CP0|SMS| normal mode, fcpu=fosc/2 faut-il le faire???????
				//Initialisation et définition du protocole de transmission/reception du st7

	SCIBRR = 0x00;				//pour le st7 seul on utilise les registres etpr et erpr
	SCICR1 = 0x00;
	SCICR2 = 0x0C;					//00001100 TIE|TCIE|RIE|ILIE|TE|RE|RWU|SBK
TE=1:transmitter enable  RE=1:receiver enable
	SCISR = (SCISR|0x20);		//initialise le fameux bit pour bien recevoir à 0


	SCIERPR = 0x07;//0x08;				//fcpu/(16*ETPR) pour le ST7 seul, on  utilise
0x08
	SCIETPR = 0x07;//0x08;

}
```

voilà l'initialisation plaquette :

```
void initst7_plaq(void)
{
	//PEDDR = 0x00;
	//PEOR = 0x00;

	MISCR1 = 0x18;				//IS11|IS10|MCO|IS21|IS20|CP1|CP0|SMS| normal mode, fcpu=fosc/2 faut-il le faire???????
						//Initialisation et définition du protocole de transmission/reception du st7

	SCIERPR = 0x00;
	SCIETPR = 0x00;

	SCIBRR = 0xC0;				//11000000;	// Baud Rate Generator: SCP1|SCP0|SCT2|SCT1|SCT0|SCR2|SCR1|SCR0
					// vitesse transmit = Fcpu/(32*PR*TR)   Fcpu=Fosc/2=16MHz/2 PR=13, TR=1 RR=1
					// 1100 0000 => 19 200 bauds
	SCICR1 = 0x00;
	SCICR2 = 0x0C;					//00001100 TIE|TCIE|RIE|ILIE|TE|RE|RWU|SBK TE=1:transmitter enable  RE=1:receiver enable
	SCISR = (SCISR|0x20);		//initialise le fameux bit pour bien recevoir à 0


}
```

V programmation ST7 – principes généraux

Chers lecteurs, ce projet est avant tout un projet d'informatique. Vous allez donc passer énormément de temps à étaler du code sur les touches de votre clavier. Il vous faudra donc vous (re)familiariser avec les bases de la programmation et du kit de développement ST7.

Nous programmons essentiellement en langage C. Nous utilisons également de l'assembleur car certaines instructions en C ne sont pas conformes aux normes du compilateur que nous utilisons. Il vous faudra également revoir l'initialisation des ports d'un ST7 et savoir utiliser les registres à bon escient.

La Mémoire nous a posé un gros problème car nous tournons au maximum à 8MHz. Par ailleurs la mémoire RAM disponible est seulement de 256 octets. On a quand même une ROM de 16 Ko suffisant pour les codes que vous implanterez. Ces contraintes impliquent que vous devez faire des économies:

Déclarez le plus de variables globales et déclarez les tableaux en constante si vous ne les modifiez pas au cours de l'exécution afin qu'ils soient implantés dans la ROM.

comme par exemple **const unsigned char L11[]={'L','1',1,1}** qui permet d'allumer la led rouge de la caméra. En effet toutes vos variables seront dans la RAM et vous n'en avez pas assez.

N'utilisez des structures de pointeurs que dans les déclarations des procédures. En effet le ST7 ne dispose pas de structures d'allocations dynamiques. C'est en vain que nous avons essayé des définitions comme typedef struct {}.
Eviter d'inclure trop de bibliothèques car elles prennent beaucoup de place lors des compilations.
Nous également refaire la bibliothèque math.h pour pouvoir utiliser des fonctions telles que sin, cos, arctan, …
Vous trouverez plus loin dans ce rapport des calculs de développements limités pour vous aider dans les fonctions indispensables.

Les fichiers.ls vous donnent le mapping de l'utilisation de la mémoire. Servez-vous en en pour avoir une idée des codes à optimiser.

Les registres du ST7 sont directement accessibles en incluant le fichier **#include <io72334.h>.** Pour changer leur contenu, il suffit de faire une affectation soit en décimal, en binaire ou en hexadécimale. Voici trois déclarations équivalentes:

```
SCIBRR=   0xC0;
SCIBRR=   0x11000000;
SCIBRR=   192;
```

N'oubliez pas d'utilisez des masques (ET, OU en l'occurrence) pour modifier les bonnes valeurs.

Utilisez le plus le type unsigned char car ils sont équivalents au type entier et occupe 1 octet mémoire contrairement aux int qui en prennent 2. Sachez que les unsigned char sont préconisés tant que les données que vous utiliserez n'excèdent pas 255, ce qui sera généralement le cas.

Par la suite nous utiliseront des déclarations externes de variables car nous avions saturé la RAM.

Par un souci de compréhension, les variables des fonctions qui vont suivre sont en internes, mais par la suite elles seront redéfinies en externes dans le code final.
A chaque fois que cela sera mentionné, nous vous donnerons un tableau désignant quelles sont les variables internes qui ont été remplacées par des variables externes dans les fonctions concernées.

VI Caméra : mode de fonctionnement

Après ces quelques pages d'introduction, vous allez enfin pouvoir percer le secret du fonctionnement de la caméra. Voici donc ci-dessous les codes en C que nous avons pris soin de vous commenter et de vous expliquer pas à pas pour une meilleure compréhension. Sachez cependant qu'une bonne connaissance dans le fonctionnement et l'utilisation de la caméra pourrait s'avérer utile, voire indispensable, pour une compréhension parfaite et totale.

D'autre part, quelques explications sommaires ne seront pas de trop :

° La caméra envoie toujours des valeurs précédées d'une lettre sous formes de trames : C 40 12 …
° La caméra marque toujours ses fins de trames par un retour chariot pour passer à la ligne suivante : M 10 20 5 10 15 30 5 75 \r
Il est donc important d'en faire de même lorsque vous lui envoyez une commande pour que cette dernière soit validée.
Ex : const unsigned char L12[]={'L','1',' ','2',0x0D};
Voici la commande qu'il faut envoyer pour allumer la diode rouge de la caméra.
Remarquez que le tableau se termine par 0x0D qui correspond à '\r' (retour chariot) en ASCII.

° La caméra inverse toutes les images qu'elle perçoit, comme si l'image renvoyée était celle reflétée par un miroir.

VI – 1 Fonctions conversion

Ces fonctions de conversion permettent de traduire la trame envoyée par la caméra en convertissant les valeurs dans la chaîne de caractères en tableau d'entiers, et inversement. La fonction saute la première lettre du début de trame pour ne récupérer que des nombres.
Ex : vous recevez « C 20 12 40 60 » qui correspond aux coordonnées du cadre contenant des balles. Il vous faut donc récupérer ces valeurs ASCII pour pouvoir travailler par la suite avec un tableau d'entiers correspondant à T[20][12][40][60].

Voici donc la fonction qui permet d'effectuer ce travail :

```
void addidas (unsigned char motst7[], unsigned int mot[], int longueur)
{
        boucle=0;
        while(((motst7[boucle] < 0x30) || (motst7[boucle]>0x39)) && (motst7[boucle] != 0x0D))
          {boucle++;}

    //on enleve les espaces et la première lettre de début de trame
    ceinture = 0;

        do
        {
                mot[ceinture] = 0;
                do
                {
```

```
                    mot[ceinture] = 10*mot[ceinture] + opticien(motst7[boucle]);
                    boucle++;

        }
        while((motst7[boucle] != 32) && (motst7[boucle] != 0x0D));// un caractère qui n'est pas séparé d'un
autre par un espace appartient au même entier
         do
         {boucle++;}
         while((motst7[boucle] == 32) && (motst7[boucle] != 0x0D));// on passe les espaces
        ceinture++;
        }
        while((ceinture <= longueur) && (tableau_mvt[boucle] != 0x0D));// condition de fin sur retour chariot
ou la longueur à convertir.
}
```

Remarque : opticien est la fonction qui correspond à la fonction en C atoi (array to integer).

La fonction qui suit fait exactement l'inverse de la fonction précédente, à savoir, elle transcrit un tableau d'entier en chaîne de caractères en incluant un espace entre chaque entier et un retour chariot à la fin de la chaîne.

```
void addidas_bar (unsigned char motst7, unsigned int mot, int longueur)
{
        boucle = 0;
        for(ceinture = 0;ceinture<longueur;ceinture++)
    {
        para3 = mot[ceinture];
        temp1 = para3%10; // on récupère les unités
        para3 /= 10;
        temp2 = para3%10; // on récupère les dizaines
        para3 /= 10;
        temp3 = para3%10; //  on récupère les centaines
                if (temp3>0)
                {
                        motst7[boucle] = temp3 + 0x30; on convertit en ASCII en ajoutant 30 en hexadécimal.
                        boucle++;
                        motst7[boucle] = temp2 + 0x30;
                        boucle++;
                        motst7[boucle] = temp1 + 0x30;
                        boucle++;
                        motst7[boucle] = ' ';
                }
                else if (temp2>0)
                {
                        motst7[boucle] = temp2 + 0x30;
                        boucle++;
                        motst7[boucle] = temp1 + 0x30;
                        boucle++;
                        motst7[boucle] = ' ';
                }
                else
                {
                        motst7[boucle] = temp1 + 0x30;
                        boucle++;
                        motst7[boucle] = ' ';

                }
```

```
                boucle++;

        }
        motst7[boucle - 1] = 0x0D; // on termine le tableau par un retour chariot.

}
```

VI – 2 La fonction SEND

Cette primitive permet d'envoyer les commandes à la caméra par le ST7 sous forme de paquet d'octets. Comme il a été vu précédemment, la caméra fonctionne en ASCII, il faut donc lui envoyé chaque mot, caractère par caractère. Par exemple, pour envoyer « TC 180 240 16 16 16 16 » , vous devez écrire dans le registre SCIDR du port série du ST7 l' octet correspondant à chaque caractère de la chaîne. i .e : vous devez envoyer « T » puis « C » puis « » ,puis « 1 » puis « 8 », etc…

Pour cela, nous avons utilisé et amélioré la fonction de l'année dernière ci-après :

```
// *************  fonction send ********************
void send(unsigned char commande[])
{
    boucle = 0;
        while(commande[boucle]!=0x0D)
        {
                do
                        { regt = SCISR;}
                while((regt&0xC0) == 0x00);        // acess to SCISR register
                SCIDR = commande[boucle];        // write byte to send to SCIDR; clearing of TDRE bit
                tempo(Temp);
                boucle++;
        }

                do
                        { regt = SCISR;}
                while((regt&0xC0) == 0x00); // acess to SCISR register
                SCIDR = 0x0D;
                para1 = Temp;                // write byte to send to SCIDR; clearing of TDRE bit
                tempo();


}
```

Il faut espacer d'une petite temporisation chaque caractère envoyé, pour éviter les collisions entre les octets. Nous utilisons donc une fonction tempo(unsigned int l) avec k=170.

```
/  ************tempo : 170 correspond à peu près au temps qu'il faut pour envoyer un octet à 19200 b/s
void tempo(unsigned int l)            //la tempo est tantôt de 200, tantôt de 150. 150 partout ne
marche pas. avec 200? faut voir
{
        unsigned int k;
```

```
        for (k=0;k<l;k++)
                k=k;
}
```

N.B : il est important de savoir qu'entre chaque envoie il s'avère parfois judicieux de mettre une temporisation de 60 voire 80.

VI – 3 La fonction RECEIVE

Cette primitive permet de recevoir les données émises au ST7 par la caméra sous forme de paquet d'octets. Au niveau du hard, ces octets sont également envoyés sur le port série, et arrive aussi dans le même registre SCIDR. (cf III-2)

Le principe de cette fonction fait que chaque octet reçu est immédiatement placé dans une case d'un tableau de caractères. On obtient donc en sortie une chaîne de caractère qu'il ne reste plus qu'à traiter.

```
void receive(unsigned char mot[])
{
        ceinture = 0;
        for(boucle = 0;boucle<5;boucle++) // on passé le "ACK" envoyé en début de chaque trame
        {
        do
                {regt = SCISR; }
        while((regt&0x20) == 0x00);
        tempo(Temp);
        }
        regt1 = SCIDR;
        para1 = 0;
        while ((regt1 != 0x0D) // on attend un retour chariot
    {
                do
                {
                regt = SCISR;
                regt1 = SCIDR;
                para1++;
                }

                while(((regt&0x20) == 0x00) && (para1 < 32000)); // on vérifie qu'il y a bien quelque chose
                mot[ceinture] = regt1;                          // dans le registre
                ceinture++;
        }
//      mot[ceinture] = '\0';
}
```

VI-4 MODE RM 3

Les ACK ET NCK d'acquittement ou de non acquittement envoyés par la caméra sont supprimés. Les données envoyées par la caméra sont directement le code ascii des coordonnées. Par exemple pour envoyer la coordonnée 37 vous aurez "%" au

lieu de "3" puis "7". Ainsi vous vous rendez compte que la fonction de conversion s'avère dans ce cas non indispensable surtout que vous économisez un octet sur deux.

VI-5 MODE RM 7

Pour mieux faire, vous pouvez configurez les entrées de la caméra comme précédemment. C'est le mode RM 7. Il est beaucoup plus efficace mais plus difficile à manipuler. Vous ne pouvez plus vous permettre des commandes du genre send("L1 0") pour allumer une diode ou encore send ("RS") pour réinitialiser la caméra.

D'abord les retours chariot sont supprimés et les trames d'information de la caméra qui étaient séparés par des caractères return sont maintenant séparées par des caractères 255. Lorsqu'une coordonnée vaut 255, la caméra envoie 254.

C'est ce mode que nous avons finalement adopté et vous trouvez de plus amples renseignements à ce propos dans la documentation de la caméra et dans la partie développement de la deuxième stratégie. Les codes d'envois et de réception sont décrits dans la partie communication.

VI-6 MODE Poll Mode PM

Lorsqu'elle est interrogée, la caméra renvoie sans arrêt des suites de trames d'informations. Afin d'éviter ces envois inutiles et pénalisants, le mode pm permet de demander à la caméra ne n'envoyer qu'une trame à chaque interrogation.

VI-7 Mode paquet de données MM

Il permet de spécifier de type de paquet d'information que vous désirez: paquet de type C, M ou N.

La trame de type C renvoie les valeurs du cadre correspondant à la couleur recherchée par la caméra. La trame de type M renvoie en plus au début les masses moyennes en X et en Y du cadre recherché.

VII Développement de la première stratégie

Notre première stratégie était de trouver tout simplement un ballon avec la caméra, et d'identifier ensuite ses coordonnées par rapport au robot.

Pour cela, nous utilisons une fonction récurrente qui permet de trier tous les éventuels ballons en recadrant sa vision, jusqu'à ne voir plus qu'un seul ballon.

Afin de savoir où se trouvent les ballons on fait des "Set Windows" à l'aide de la commande SW de la caméra. Sur une image de 80x143, cette commande permet de faire un recadrage sur une zone précise, par exemple pour pouvoir recadrer l'image autour d'un ballon.

Ensuite, il faut traquer la couleur que vous recherchez afin de savoir si un ballon est ou n'est pas sur la petite zone de recadrage. On se sert pour cela de la fonction TC de la caméra.

VII – 1 traquer une couleur

Le processeur de la caméra possède une fonction qui permet de chercher la couleur que l'on souhaite trouver. La fonction renvoie les coordonnées des deux sommets du cadre dans lequel se trouve la couleur recherchée (cf datasheet). Les sorties de la caméra permettant d'obtenir directement les coordonnées de la couleur qu'on recherche, ces derniers pourront directement être transmis au microcontrôleur afin d'orienter le robot.

Ainsi, dans notre cas, le ballon recherché était de couleur rouge, mais dont la composante rouge variait entre 160 et 240. Les deux autres composantes demeuraient fixes à la valeur 16.

On devait donc envoyer la commande « TC 160 240 16 16 16 16 »

VII – 2 Où sont les ballons ?

Voici la boucle que nous avons incorporée dans la plupart des fonctions pour trouver les balles.

Attention: ici, il y a un changement de variable interne d'entrée/sortie de la fonction receive !

A la place de …	Nous utilisons directement dans le code
Unsigned char mot[]	Unsigned char tableau_mvt[]

```
a = 0;
do
{
        send(TCred);  // envoie la couleur de la balle (TCred est la chaîne de caractères correspondant)
        tempo(Temp);
        receive();
```

```
            tempo(Temp);
            send(alive); //   remet la caméra en service.
            tempo(20*Temp);
            a ++;
    }
    while ((a < 7) && (tableau_mvt[2] == '0')); // on fait plusieurs fois le test.
```

Le send(TCred) envoie la commande « TC 180 240 16 16 16 16 » qui correspond à la couleur d'une balle rouge. (Paramètre peu fiable selon la luminosité). Par la suite, la caméra renvoie les valeurs du cadre correspondant à la couleur demandée dans une trame de type M :
« M 60 100 45 80 75 120 8 50 ».
Il faut ensuite convertir ces données en entier grâce à la fonction de conversion qui met chaque valeur dans une case du tableau. Comme la plupart du temps, ce tableau était modifié, nous conservions les valeurs originales dans un autre tableau de référence, noté ref, grâce à la fonction trans :

```
void trans(void)
 {
        for(boucle = 0;boucle < 6;boucle++)
        {ref[boucle] = mot[boucle];}
 }
```

Nous travaillions également tout le temps avec des trames de type M, alors qu'il nous arrivait parfois de ne pas utiliser les deux premiers arguments. Nous utilisions alors la fonction trans2 qui permet d'écraser les deux coordonnées des masses moyennes.

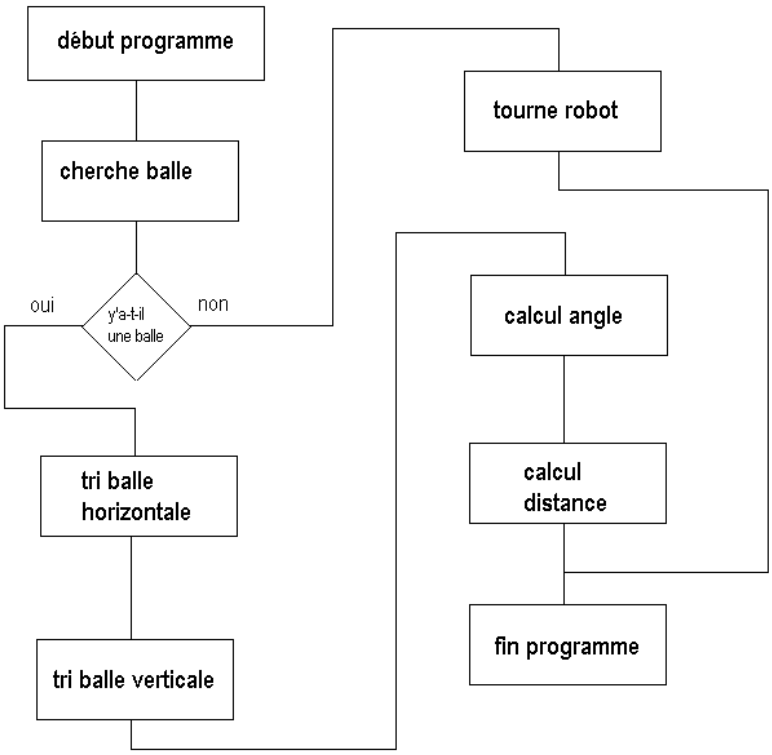
```
void trans2(void)
{
        for(boucle = 0 ; boucle < 4 ; boucle++)
        {mot[boucle] = mot[boucle + 2];}
}
```

VII – 3 Explication du contenu de la fonction ramène.

Cette fonction est incluse dans le fichier bouh.c et représente une partie de l'ia concernant la caméra. Nous allons voir d'abord l'élaboration de la stratégie traduite en C, puis les sous-fonctions qui nous ont servi d'interface avec la caméra, et enfin les calculs d'angle et de distance du ballon par rapport au robot.

Pour un souci de commodité, nous avons mis toutes les variables en interne. Sachez cependant que pour un problème de mémoire RAM tout le code a été remanié en mettant les variables en externe, nous vous montrerons comment.

La fonction ramène(void) est la fonction appelée par le programme général pour ramener les coordonnées de la balle la plus proche. A défaut de trouver une balle, le robot effectue une rotation sur lui-même puis relance la fonction.



VII – 3 – 1 la fonction triballe

Explication de la fonction triballe :

Cette fonction permet de trier les balles à la verticale et à l'horizontale pour n'obtenir à la fin qu'une seule balle.

Avant l'appel de la fonction la caméra perçoit cette fenêtre :

Le problème étant que la caméra n'est pas capable de distinguer l'une ou l'autre balle simplement en recherchant la couleur.
Il nous faut donc découper la vision en deux. Pour cela, nous travaillons sur la trame renvoyée. Par exemple, si nous recevons la trame M 60 100 45 80 75 120 8 50, nous savons que les coordonnées des deux sommets du cadre sont (45,80) et (75,120) (cf VI – 7 et annexe 2). La variance en Y (120 – 80) est trop grande pour correspondre qu'à une seule balle. Nous allons donc diviser le cadre en

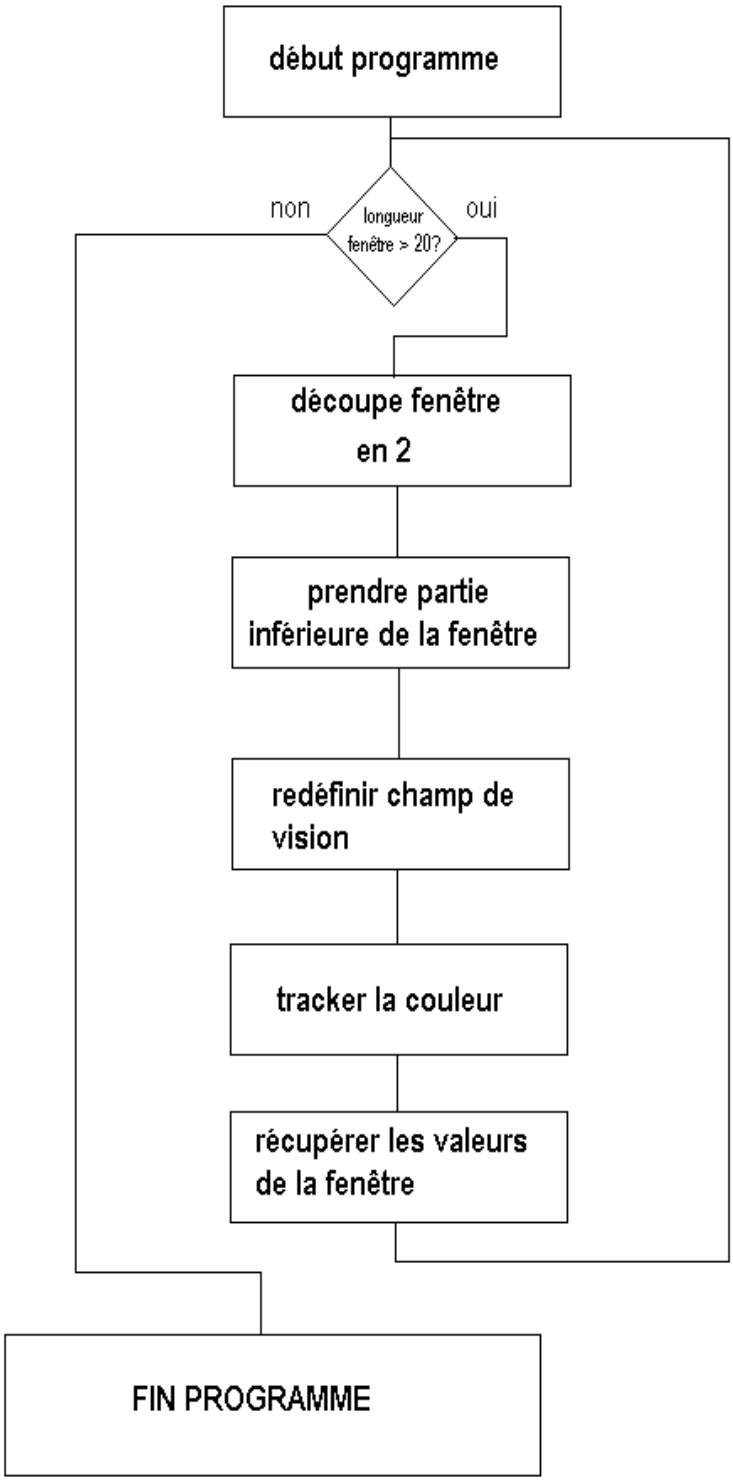
utilisant la masse moyenne de telle sorte que ses coordonnées en Y seront 80 et 100. Puis nous redéfinissons la fenêtre en envoyant un SW (set window).

Après l'appel de la fonction nous obtenons donc l'image ci-contre.

Par la suite, nous faisons le test sur la largeur du cadre pour savoir s'il correspond à la largeur d'une balle. Cette opération se répète jusqu'à obtenir un cadre de largeur équivalente à la largeur d'une balle. Une fois que le ST7 sort de la boucle du programme nous savons que nous travaillons sur la ligne correspondant aux balles les plus proches.

En ce qui concerne le code ci-dessous, les variables d'entrée/sortie des fonctions de conversion que vous connaissez bien (addidas() et addidas_bar()) ont été déclarées en externe.

A la place de …	Nous utilisons directement dans le code
Unsigned char motst7[]	Extern Unsigned char tableau_mvt[]
Unsigned int mot[]	Extern Unsigned int mot[]
Int longueur	Extern unsigned int para2


début programme
non
longueur fenêtre > 20?
oui
découpe fenêtre en 2
prendre partie inférieure de la fenêtre
redéfinir champ de vision
tracker la couleur
récupérer les valeurs de la fenêtre
FIN PROGRAMME

```
//////////////////////////////////////
//////tribal

        _asm("tatoo3:");

        if ((mot[5] - mot[3]) > 20)    //coupage en deux vertical. On fait la récurrence tant qu'on n'a pas une balle
        {
        mot[5] = mot[1]; // on prend la partie inférieure du cadre
        trans2(); // on sauvegarde les valeurs du cadre
        para2 = 4;
        addidas_bar();     // on remet les valeurs dans un tableau de caractères (traduction des entiers en ASCII)
        boucle = 17;
        do
        {tableau_mvt[boucle + 3] = tableau_mvt[boucle];
                boucle--;}
        while (boucle > 0);
        tableau_mvt[3] = tableau_mvt[0];
        tableau_mvt[0] = 's';
        tableau_mvt[1] = 'w';
        tableau_mvt[2] = ' ';  // on écrit dans tableau_mvt la commande à nvoyer :tableau_mvt[]= « SW … »
         send(tableau_mvt); // on redéfinit les dimensions du cadre à la caméra
        tempo(100);
        send(alive);
        tempo(20*Temp);
        send(tableau_mvt); // on le refait une fois, on n'est jamais trop prudent…
         tempo(100);
        send(alive);
        tempo(60*Temp);
        a = 0;
        do
        {
                send(TCred);
                tempo(Temp);
                receive();
                tempo(10*Temp);
                send(alive);
                tempo(20*Temp);
                a ++;
        }
        while ((a < 7) && (tableau_mvt[2] == '0')); // on test si on a une balle
        if (tableau_mvt[2] == '0')
        {_asm("JP pa_balle");}
        else
        {para2 = 6;
        addidas(); // on traduit les valeurs du cadre en entier
        _asm("JP tatoo3");}
        }
```

Une fois que la fenêtre a été coupée en horizontal, on va travailler sur la partie inférieure du cadre. (C'est-à-dire sur les balles les plus proches).

Admettons que l’on soit dans le cas suivant :

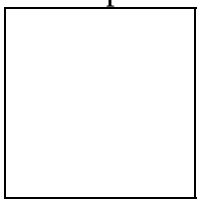
Nous devons donc trier les deux balles pour n’obtenir plus qu’une seule balle à la fin. Pour cela, nous lançons la suite du code qui permet de trier les balles dans le sens vertical.

Nous obtenons donc à la fin :

De même que précédemment, le programme s’exécute tant que la longueur du cadre ne correspond pas au moins à la longueur d’une seule balle. Ainsi l’image est découpée plusieurs fois, et la caméra relance à chaque fois la recherche de balle dans son nouveau cadre.

Cette opération, bien que longue et fastidieuse, gagne en rapidité à chaque fois qu’elle est appelée, car le cadre voit ses dimensions diminuer à chaque appelle.

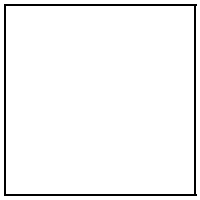
Un schéma explicatif vous est présenté sur l’autre page suivi du code associé.



```
_asm("tatoo:");
para4 = mot[4] - mot[2];                    // on calcule la variance dX.
if (para4 > 15)                                     // on fait la récurrence tant qu'on ne tombe pas sur
                                                    // une seule balle.(dX<15)
{       mot[4] = mot[2] + para4 / 2 + 1;    // on travaille sur la première moitié du cadre visionné
        trans();  // on sauvegarde les valeurs du cadre de début dans la variable ref[]

        ////////////////////////////////////////////////////////////////////////
```

```
		///////cherche
		_asm("tatoo2:");
		trans2(); // on décale le tableau pour écraser les entiers correspondant aux masses moyennes.
		para2 = 4;
		addidas_bar(); // on redéfinit le cadre
		boucle = 17;
		do
		{tableau_mvt[boucle + 3] = tableau_mvt[boucle];
			boucle--;}
		while (boucle > 0);
		tableau_mvt[3] = tableau_mvt[0];
		tableau_mvt[0] = 's';
		tableau_mvt[1] = 'w';
		tableau_mvt[2] = ' '; // on écrit dans tableau_mvt[]=  « SW … »
		send(tableau_mvt); // on envoie la commande.
		tempo(100*Temp);
		send(alive);
		tempo(20*Temp);
		send(tableau_mvt); // on ré envoie la commande.
		tempo(100*Temp);
		send(alive);
		tempo(10*Temp);

		temp1 = 0;
		do
		{
			temp1++;
			send(TC1); // on envoie plus que l'instruction « TC »
			tempo(Temp);
			receive();
			tempo(10*Temp);
		} while ((temp1< 7) && (tableau_mvt[2] == '0')); // on fait le test au moins 7 fois.

		para2 = 6;
		addidas();  // on traduit les valeurs en entier
		if (mot[0] == 0)
			{ceinture = 0;}
		else
			{ceinture = 1;}
	////////fin cherche
	//////////////////////////////////////////

		if (ceinture == 1)		//on lui demande si elle voit la balle la plus proche.
			{_asm("JP tatoo");
		}
		else
		{
			ref[2] = ref[4];
			ref[4] += para4/2;		// on travaille sur la seconde moitié du cadre visionné
			mot[2]= ref[2];
			mot[3]= ref[3];
			mot[4]= ref[4];
			mot[5]= ref[5];
			if (ref[4] < 81)
			{_asm("JP tatoo2");}

		}
	}
// on définit les nouvelles dimensions du cadre et on relance la fonction jusqu'à voir une seule balle.
```

/////fin tribal

En appelant les deux fonctions de tri vertical et horizontal, on arrive à obtenir à la fin un cadre centré sur une seule balle. Ceci nous permet par la suite de connaître exactement le centre de la balle est de pouvoir travailler sur un point. 
Ci-dessus le cadre final perçu par la caméra lors du dernier appel de al fonction. La caméra nous renvoie donc la trame équivalente : « M 30 50 25 45 35 55 8 90 »
Par la suite, nous savons que les coordonnées en X et Y des masses moyennes (respectivement 30 et 50) correspondent exactement au centre de la balle. Nous pouvons donc ainsi calculer les coordonnées réelles de la balle sur le terrain.

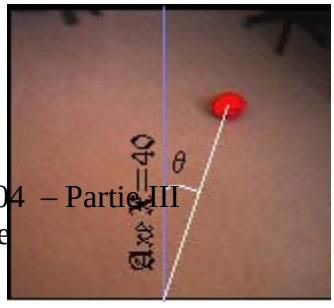
Le fait d'avoir créé cette fonction en récursif permet de gagner du temps et de la place dans le code. Néanmoins elle présente un inconvénient lors du renvoie des coordonnées. En effet, selon la position de la balle par rapport à la caméra, le tri a tendance à couper la balle en deux et à renvoyer un point de l'extrême bout de la balle. Ceci peut poser généralement un petit problème lors des calculs d'angle, car si le point échantillonné n'est pas au centre du ballon, le robot ne sera pas exactement centré sur la balle.
A vous de voir par la suite la marge d'erreur que peut se permettre le robot lorsqu'il doit attraper l'objet en question.

VII – 3 – 2 Calcul de l'angle et de la distance

Une fois que nous avons obtenu les coordonnées du cadre comprenant un seul ballon, nous récupérons seulement les valeurs des masses moyennes correspondant au centre du cadre. (Donc logiquement, au centre de la balle.)

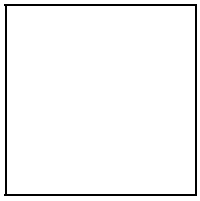
Ainsi nous travaillons finalement sur deux valeurs.
Ex : si la trame finale renvoyée est « M 15 20 7 15 23 25 8 100 » alors nous récupérons les deux valeurs X=15 et Y=20 (ne perdez pas de vue le fait que l'axe des X est inversé sur la caméra. X=15 n'est pas un point plutôt à gauche, mais plutôt à droite !)
Nous avons recentré l'origine des axes de sorte qu'un ballon qui se trouve dans l'axe du robot (X=40) soit centré en X=0. Il suffit juste de faire une translation d'axe : X=X-40.
Par la suite, ceci nous permettra de savoir si la balle est à gauche (X > 0) ou si elle se trouve à droite (X < 0).

Soient X et Y les coordonnées de la balle dans le plan de la caméra (photo ci-contre). Alors on a la relation :

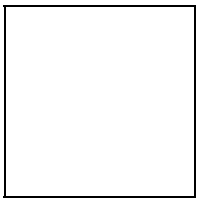
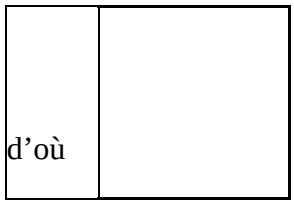


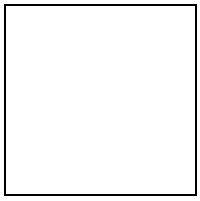
$\text{Tan}(\theta) = (X - 40)/Y$

Cependant, il faut noter la présence d'un changement au niveau de l'axe des ordonnées. Car, ce n'est pas parce qu'une balle se trouve collée contre le bas de l'image qu'elle se trouve collée aux pieds du robot. En effet, il existe un espace entre le robot et la première balle vue par la caméra, qui ne se trouve pas dans le champ de vision de cette dernière. Cet espace est en fait dans l' « angle mort de la caméra ». Donc, de la même manière qu'il a fallu recentrer l'axe des abscisses, il va falloir décaler l'axe des ordonnées de la manière suivante :



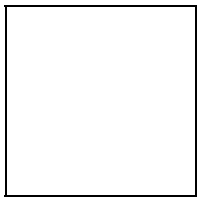
α est l'angle d'inclinaison de la caméra. ψ est le ½ angle du champ de vision vertical de la caméra est vaut 17°. H est la hauteur de la caméra sur le robot. L est la longueur de l'espace invisible pour la caméra et qu'il faut rajouter à l'ordonnée pour connaître la véritable valeur en Y du ballon par rapport au robot.

On a donc  d'où 

La relation précédente devient donc 

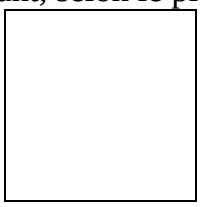
Intéressons-nous maintenant à la fonction qui permet de calculer l'angle réel : L'angle de la balle par rapport au robot est fonction de l'angle de la balle par rapport à la caméra et de l'angle d'inclinaison de cette dernière sur le robot. Nous avons donc au final trois angles :

α : Angle d'inclinaison de la caméra. Appartient au plan (XOZ)
β : Angle de la balle par rapport à la caméra. Appartient au plan (XOY)
θ : Angle de la balle par rapport robot. Appartient au plan (XOY)

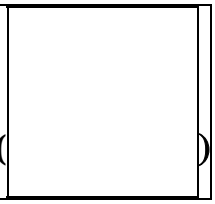


Frédéric Vallotti
Benjamin Bounous

Maintenant, selon le plan ci-dessus on a la relation :

Tan(β) =  .

L'angle final recherché est donc l'angle β**,** c'est-à-dire l'angle de la balle par rapport au robot.

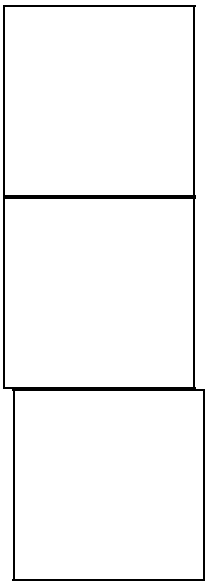
β =**arctan(**  **)**

Voici notre code associé. Attention, les fonctions trigonométriques Cos() et arctan() sont des fonctions dont la variable **<u>externe</u> a** est en entrée **ET** en sortie.

```
a = mot[0]; // on récupère l'abscisse X du point
a -= 40;  // on recentre en X
b = mot[1]; // on récupère l'ordonnée Y du point
b += H/tan_angle;// longueur (en pixel?) manquant à la latitude.
b = a/b;
a = angle; // la variable « angle » est en fait l'angle α.
cos();
a = b / a; //a=tan(theta)/cos(angle);
arctan();  // on récupère dans la variable a la vrai valeur de l'angle β.
```

Calculons maintenant la distance de la balle par rapport au robot.
Pour cela, nous avons utilisé un moyen plus simple mais moins précis : le polynôme de Legendre. Le fait de perdre en précision n'est pas très grave dans le sens où, si le robot va un peu plus loin que la distance exacte il prendra quand même la balle.

Le polynôme de Legendre se calcule empiriquement. C'est-à-dire qu'il suffit de mettre le ballon à différentes distances D du robot (mais dans le même axe) et de regarder sur la caméra quelles sont les valeurs en Y équivalentes.
Il ne reste plus ensuite qu'à résoudre le système en fonction de D (distance réelle) et Y(affixe du point dans le plan de la caméra) :



Une fois le système résolu, vous avez trouvé les valeurs de a,b et c. il suffit donc de rentrer le polynôme de Legendre dans le code est de remplacer chaque fois que nécessaire la valeur en Y de la caméra pour trouver la valeur de la distance.
Voici le code associé :
Attention : ici, la variable d'entrée de la fonction tempo a été redéfinit en externe

A la place de	Nous utilisons directement dans le code
Unsigned int l	Extern unsigned int para1

```
void stance(void)
{
        send(L11); // allume la diode rouge
        para1 = 7*Temp;
         tempo();
         send(alive);
         tableau_mvt[compteur_mvt] = '3'; // donne au robot l'ordre d'avancer
         cos();
         a = (43 + 0.3 * b + 0.002 * b * b) / a;  // 25.7 0.1898 0.00296 -> polynôme de Legendre
         para3 = a;                                // on divise par le cosinus de l'angle car le ballon
         tableau_mvt[compteur_mvt+1] = para3; // n'est pas forcément dans l'axe du robot.
         tableau_mvt[compteur_mvt+2] = '0';
         send(L10);
         b = 0;
}
```

VII – 3 – 3 la fonction alerte

Cette fonction permet de faire clignoter la led rouge de la caméra du nombre de fois que vous le désirez. Cette fonction anodine et très simple à écrire est en fait très pratique lors du debug de votre programme. Car lorsque vous ne travaillez pas sur la plaquette de simulation, mais directement sur le ST7 vous ne pouvez pas voir votre code, et donc, vous ne pouvez pas faire du pas à pas. Mettre des « alerte » un peu partout dans vorte code vous permet de savoir où en est l'exécution du code.

```
//******fonction qui sert juste à faire clignoter la diode rouge de la caméra
void alerte(unsigned int j)                    //nombre de pulse désirés en entré
{
        for(temp1=0;temp1<j;temp1++)
        {
                send(L11); // on allume la led
                para1 = 10000;
                tempo();
                send(L10); // on éteint la led
                para1 = 10000;
                tempo();
        }

}
```

VIII Description des programmes secondaires : deuxième stratégie

Dans la suite du rapport, nous allons vous exposer les fonctions de second niveau que nous avons hélas pas eu le temps de mettre en pratique. Certaines de ces fonctions ont été créées à des fins stratégiques, de manière à gagner du temps lors des matchs, d'autres ont été conçues de manière à remplacer certaines parties du robot qui se trouvaient être défectueuses. Cependant, nous avons cruellement manqué de temps pour pouvoir utiliser toutes ces fonctions, bien que simulées par le ST7 elles se sont révélées opérationnelles.

VIII – 1 la fonction IA

La fonction IA contient en fait le code de l' « intelligence artificielle » du robot. C'est cette fonction que nous lancions après l'initialisation de la caméra et du ST7, et qui permettait au robot de « réfléchir » à la trajectoire à effectuer pour capturer les balles.

Pour communiquer avec l'asservissement des moteurs du robot nous devions remplir un tableau qui correspondait à la trajectoire que nous voulions qu'il réalise.
Nous écrivions donc dans le tableau les valeurs correspondant aux angles et distances vus précédemment, puis un autre programme se chargeait de lire ces données et de faire bouger le robot en conséquent.
Nous remplissions le tableau comme suit :

Extern unsigned char tableau_mvt[] : tableau déclaré en externe dans lequel la trajectoire du robot est implantée. Pour cela, il faut écrire la commande dans toutes les cases paires, et les arguments dans les cases impaires :

Tableau_mvt[2n] = « commande »

Commande = '1' -> rotation gauche

Commande = '2' - > rotation droite

Commande = '3' -> avance

Commande = '4' -> recule

Commande = '0'-> fin de trajectoire

Tableau_mvt[2n+1] = argument

Ex : tableau_mvt[] = { ‘2’ , 45 , ‘3’ , 15 , ‘1’ , 10 , ‘4’, 21 , ‘0’ } correspond à la trajectoire suivante :

Rotation droite de 45° ---- Avance de 15 cm ---- Rotation gauche de10° ---- Recule de 21 cm---- fin de trajectoire.

Un code de recadrage a également été mis au point dans le cas où le robot ne voyait pas de balle sur le terrain. Ce code consistait à faire une rotation de 30° à gauche la première fois et de relancer la procédure de détection de balle. Si les balles n’étaient toujours pas visibles par le robot, alors celui-ci faisait cette fois-ci une rotation de 60° à droite, et continuait cette rotation jusqu’à ce qu’il aperçoive une balle.

```
//////////////////////
//      CODE D'EVITEMENT
//////////////////////

_asm("pa_balle:");
  if (b == 0) // b=0 : premier test.
  {
        tableau_mvt[0] = '1'; // tourne à gauche
        tableau_mvt[1] = 30; // 30°
        tableau_mvt[2] = '0'; // fin trajectoire
        b++;
  }
  else
  {
        if (b < 7)  // b>0 : c’est au moins la deuxième fois que le robot ne détecte pas de balle
        {
              tableau_mvt[0] = '2'; // rotation à droite
              tableau_mvt[1] = 60; // 60°
              tableau_mvt[2] = '0';  // fin trajectoire
              b++;
        }
        else // le robot a tourné sur lui-même 7 fois, on le fait avancé
```

```
                {
                        tableau_mvt[0] = '3'; // avance
                        tableau_mvt[1] = 30; // 30 cm
                        tableau_mvt[2] = '0'; // fin trajectoire
                        b = 0;
                }
        }
        compteur_mvt = 0;
        while (tableau_mvt[compteur_mvt] !='0')
        {
                mouvement2();
                compteur_mvt = compteur_mvt +2;
        }

        _asm("jp do_it");

      //////////////////////
//      FIN DE CODE D'EVITEMENT
        //////////////////////
```

VIII – 2 la fonction calibrage

La caméra est un appareil très sensible à la lumière ambiante. Ainsi, les paramètres qui définissent les composantes RVB de la balle sont très variables, car dépendants de la luminosité de la pièce. C'est pour cela que nous avons imaginé un code qui permettait de pallier à ce genre de problème en laissant la caméra calibrer elle-même la couleur de la balle, plutôt que de lui imposer une couleur constante à chaque fois. Grâce à la fonction de la caméra GM (get mean) qui permet de récupérer les composantes RVB de l'image ainsi que leur déviance, on pouvait adapter la couleur de l'image en mettant les valeurs récupérées en paramètres d'entrée des fonctions. Cette fonction n'a jamais été utilisée sur le robot, car trop longue à vérifier.

Voici le code associé :

```
void kalibre(void)
{
//      unsigned int mot[6];
//          unsigned int ref[6];
            unsigned int courant[3];
//          unsigned int couleur[6];
        int i;
//12 int
//0 char

            mot1[0] = 255;
            mot1[1] = 0;
            mot1[2] = 255;
            mot1[0] = 0;
            mot1[1] = 255;
            mot1[2] = 0;


            for (i=0;i<12;i++)
            {
            send(sw_init_Klibre);
            para1 = 10*Temp;
            tempo();
            send(GM);
            para1 = Temp;
            tempo;
            receive ();
            addidas(ref,3);


            para4 = 0;
            para2 = 2;
            para3 = 1;
```

```
do
{
        do
        {
                mot[0] = para2;
                mot[2] = para2;

                mot[1] = para3;
                mot[3] = para3;
                addidas_bar(mot,4);
                do
                { boucle++; }
                while (tableau_mvt[boucle] != 0x0D);
                //boucle++;
                do
                {
                        tableau_mvt[boucle+3] = tableau_mvt[boucle];
                        boucle--;
                }
                while(boucle >= 0);
                tableau_mvt[0] = 's';
                tableau_mvt[1] = 'w';
                tableau_mvt[2] = ' ';
                send (tableau_mvt);
                para1 = 10*Temp;
                tempo();
                para1 = Temp;
                send(GM);
                tempo();
                receive(tableau_mvt);
                addidas(courant,3);
```

```
if ((courant[0] - seuil_red < ref[0]) && (courant[0] + seuil_red > ref[0]) &&
(courant[1] - seuil_green < ref[1]) && (courant[1] + seuil_green > ref[1]) && (courant[2] - seuil_blue <
ref[2]) && (courant[2] + seuil_blue > ref[2]))
// le changement n'est pas assez franc!!
                {
                }
                else    // le changement est suffisant
                        // on indique la discontinuité
                        // on sauvegarde les valeurs
                {
                        para4++;
                        if (mot1[2] < para2)
                        {mot1[2] = para2;}
                        if (mot1[0] > para2)
                        {mot1[0] = para2;}



                        if (mot1[0] > courant[0])
                        { mot1[0] = courant[0]; }
                        if (mot1[2] > courant[1])
                        { mot1[2] = courant[1]; }
                        if (mot1[4] > courant[2])
                        { mot1[4] = courant[2]; }
                        if (mot1[1] < courant[0])
                        { mot1[1] = courant[0]; }
                        if (mot1[3] < courant[1])
                        { mot1[3] = courant[1]; }
                        if (mot1[5] < courant[2])
                        { mot1[5] = courant[2]; }

                }
                // on change de référence!!
                ref[0] = courant[0];
                ref[1] = courant[1];
```

```
                ref[2] = courant[2];

                para2++;
        }
        while ((para2 <= 80) && ((para4%2) != 0));

        if ((para4 == 2) && (para2 < 80)) // premier front en y
        {
                mot1[1] = para3;
        }
        else
        {
                if ((para4 == 2) && (para2 == 80)) // deuxième frond (descendant)
                {
                        mot1[3] = para3;
                }
                else
                {
                        para4 = 2;
                }
        }
        para2 = 1;
        para3++;
}
while ((para3 <= 143) && (mot1[3] != 0));
```

VIII – 3 calcul de la position du robot

Nous avons été amené à créer la fonction odomètrie qu'après avoir été obligé de trouver un autre moyen pour permettre au robot de se repérer sur le terrain.

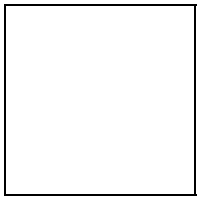
Nous avons réalisé une fonction d'odomètrie qui permet au robot de se connaître sa position sur le terrain par rapport à sa position d'origine, son angle et sa distance parcourue.
Pour cela, nous implantons un repère orthonormé (XOY) et incrémentons sa position et son orientation à chaque déplacement et rotation. Nous définissons donc trois variables qui sont :
Tortue : qui correspond à l'angle que fait l'axe du robot par rapport à l'axe X.
Position _x : qui correspond à son abscisse
Position _y : qui correspond à son ordonnée.
A chaque fois que le robot procède à un déplacement, nous appelons la fonction odométrie.
Voici le code associé :

```
void odometrie(void)
{
	if (tableau_mvt[compteur_mvt] == '1') // on test quelle est la trajectoire du robot.
	{
	tortue += tableau_mvt[compteur_mvt + 1]; // rotation gauche-> on incrémente l'angle du robot.
		if (tortue > 360)
		{tortue -= 360;}
	}

	if (tableau_mvt[compteur_mvt]== '2')
	{
		tortue -= tableau_mvt[compteur_mvt + 1]; // rotation à droite -> on décrémente
		if (tortue < 0)
		{tortue += 360;}
	}

	if (tableau_mvt[compteur_mvt] == '3') // le robot avance
	{
		a = tortue;
		a = a * (3.141592654) / 180  ;// on passe en radian
```

```
            cos();
            position_x += a * tableau_mvt[boucle]; // on projète le vecteur représentant son
                                                   //déplacement sur l'axe des abscisses.

        a = tortue;

            a = a * (3.141592654) / 180  ;
            sin();
            position_y += a * tableau_mvt[compteur_mvt + 1]; // on projète de même sur les
                                                             ordonnées


        }


        if (tableau_mvt[compteur_mvt] == '4') // on fait le même raisonnement si le robot recule
        {
            a = tortue;
            a = a * (3.141592654) / 180;
            cos();
            position_x -= a * tableau_mvt[boucle];
            a = tortue;
            a = a * (3.141592654) / 180;
            sin();
            position_y -= a * tableau_mvt[compteur_mvt + 1];
        }
        boucle++;

}
```

VIII – 4 la fonction Fire

La seconde stratégie était sensée permettre de surmonter le problème des balises, qui nous n'avons malheureusement pas pu utiliser. Le robot ne pouvant pas se situer sur le terrain, il ne pouvait naturellement pas s'orienter face aux zones de drop. La fonction fire, permettait donc de recadrer le robot face à ces zones en utilisant la fonction odomètrie.
La fonction faisait réaliser au robot une rotation, un déplacement jusque dans l'axe des drops perpendiculaire à l'axe des ordonnées, puis une nouvelle rotation pour que celui-ci se place et tire dans le bon axe.



Voici ici un exemple de déplacement pour aller tirer au but. On voit que le robot s'oriente une première fois face à l'axe des drops, puis avance ou recule selon sa latitude, et refait une rotation pour se positionner face aux drops et tirer.

Voici le code associé :

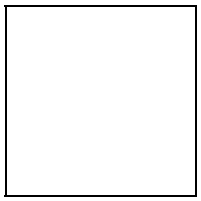
```
void fire(void)
{       temp1 = 0;

        while (tortue < 0)
        {tortue += 360;} // on travaille sur l'angle principal

        while (tortue > 360 )
        {tortue -= 360;} // on travaille sur l'angle principal

        a = position_y - 105; // 105 correspond à la latitude des drops

        b = tortue; // on met l'angle dans une autre variable qui sera modifiée

        if (b > 270) // les conditions qui suivent  empêchent de faire de trop grandes rotations
        {
             b = (b-360); //selon l'angle initial du robot, il vaut mieux faire une rotation droite
        }
        if (b > 90)
        {
              b = 180 - b; //selon l'angle initial du robot, il vaut mieux faire une rotation gauche
                 temp1 = 1;
        }
```

```
b = b + 90;
para1=b;


if (temp1 == 1)
{
        tableau_mvt[0] = '1'; // rotation gauche
        tableau_mvt[1] = para1;
}
else
{
        tableau_mvt[0] = '2'; // rotation droite
        tableau_mvt[1] = para1;
}



if ( a < 0 )
{       a = -a;
        tableau_mvt[2]='4'; // on est au-dessous des drops, on recule
        para1=a;
        tableau_mvt[3]=para1;
}
if ( a > 0 )
{
        tableau_mvt[2]='3'; // on est au-dessus des drops, on avance
        para1=a;
        tableau_mvt[3]=para1;
}

tableau_mvt[ceinture] = '2'; // on se positionne face aux drops.
tableau_mvt[ceinture+1] = 90;
tableau_mvt[ceinture+2] = '7';// on tire
tableau_mvt[ceinture+3] = '0';
```

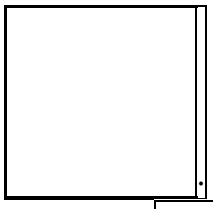
```
        tableau_mvt[ceinture+4] = '1';
        tableau_mvt[ceinture+5] = 90;
        tableau_mvt[ceinture+6] = '0';

}
```

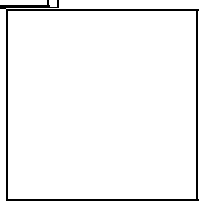
VIII – 5 la fonction 2coups

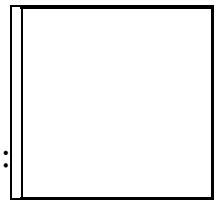
Cette fonction nous permettait de calculer la trajectoire du robot sur les deux premières balles en début de match. En effet, sachant que la position de la première balle était constante en début de chaque match, il n'était pas difficile d'orienter le robot dans sa direction. Il suffisait ensuite de calculer la position d'une éventuelle seconde balle, puis de calculer la trajectoire complète que devait faire le robot pour atteindre ces deux premières balles. Cette fonction ne pouvait évidemment être lancée qu'en début de match.



La fonction lançait une première fois la fonction ramène qui nous permettait de connaître la position de la balle de droite. La balle de gauche étant connue, on avait donc dès le départ les valeurs de γ, de a et de b. Grâce au théorème d'Al Kashi, il n'était pas difficile d'en déduire par la suite les valeurs de α, β et c. nous donnions donc au robot la trajectoire suivante :
Rotation gauche de $\gamma°$ ——— Avance de a ----- Rotation droite de (180 - β) ------- Avance de c.
THEOREME D'AL KASHI

 . Une fois que la longueur C est calculée, on en déduit la valeur de

Cos(β)= 

En simplifiant un peu, on trouve : 

Voici le code associé :

```
void etmat(void)
{
        boucle=0;
        ramene();// calcul la position de la seconde balle.
        if (tableau[2]!='0') // A-t-il trouvé une balle?
        {
              a=tableau_mvt[1]+angle_B1;
              cos();
b = dist_B1*dist_B1 + tableau_mvt[3]*tableau_mvt[3] + tableau_mvt[3]*dist_B1*a; // théorème D'Al
                                                                                  // Kashi.
              para1 = b;
             b = racine[para1];
             a =( -2*dist_B1 - tableau_mvt[3]*a ) / b;
             arccos();
             a=180 - a*180/PI
             para1 = a ;
             tableau_mvt[4] = '2';
             tableau_mvt[5] = para1;
             tableau_mvt[6] = '3';
             para1 = b;
             tableau_mvt[7] = para1;
              tableau_mvt[8] = '0';
        }
        else
        {
              tableau_mvt[4]='0'; // si il n'y a pas de 2nde balle, on ne fait rien
        }
        tableau_mvt[0] = '1'; // on prend la première balle qui est constante.
        tableau_mvt[1] = angle_B1;
        tableau_mvt[2] = '3';
        tableau_mvt[3] = dist_B1;
}
```

CONCLUSION

Frédéric Vallotti

L'élaboration du code du robot a été une bonne expérience. Je pense qu'il est intéressant de faire ce genre de projet au moins une fois durant son apprentissage scolaire, car il permet de mettre en pratique ce que l'on a appris en un temps limité. En ce qui me concerne, le projet m'a permis de pousser mes connaissances en matière de code et de ST7. J'ai pu également pu faire le parallèle entre le fonctionnent série caméra/st7 et les cours de réseau de cette année.

Pour les futurs participants, il faut savoir que le poste occupé sur la caméra est avant tout un projet de programmation et très peu d'électronique. Il faut cependant savoir faire le lien entre le côté hard du ST7, et sa programmation. Vous allez passer de longues heures à comprendre le fonctionnement série de la caméra et du st7, et vous allez apprendre comment marche un bus TTL. N'oubliez pas que tous les premiers tests effectués devront se faire sur plaquette de simulation, il vous faudra donc lire attentivement la notice pour savoir quels sont les registres du st7 à initialiser pour communiquer à la vitesse désirée. N'oubliez pas non plus de mettre des masses communes entre plaquette de simulation et port TTL de la caméra. En effet, nous avons passé de longs moments à comprendre pourquoi la caméra ne voulait pas communiquer, alors que le signal du st7 et de la caméra n'avait pas la même référence.

Pensez à faire de la place mémoire dans votre code. Le st7 n'a seulement que 256 octets de RAM disponible, et vous ne serez probablement pas les seuls à coder sur le même st7. Utilisez un maximum de déclaration en global, cela prend moins de place.

Familiarisez-vous avec la caméra sur le logiciel java prévu à cet effet. Attention, pour faire fonctionner le logiciel, la caméra doit travailler en 115 000 bauds.

Nous n'avons pas pu utiliser le servo-moteur de la caméra cette année, car nous n'avons pas eu le temps de nous pencher sur le problème. Il serait cependant intéressant pour les futurs participants d'essayer de le faire fonctionner en coordination avec l'asservissement du robot. Le principe étant que la caméra se recentre toujours sur la couleur de l'objet à chercher, le moteur fait directement le travail de tourner et de déplacer la caméra. Il faut cependant élaborer un système qui puisse rendre le chassie du robot complètement dépendant de la caméra, ce qui n'est pas forcément la meilleure solution.

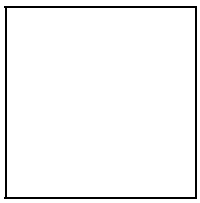
Prenez le temps de faire étape par étape, et vérifiez que la communication marche bien, avant de vous lancer dans une quelconque stratégie.

Faites des tests, des tests, et encore des tests. C'est le seul moyen d'optimiser vos lignes de code. Attention également à la vitesse du st7. Ce n'est pas parce que cela marche sur plaquette de simulation que ça marchera sur le st7. Un test pratique vaut

toujours mieux qu'un code théorique. Il vous faudra parfois remanier un peu votre code car le robot est têtu par nature et ne fait pas toujours ce qu'on lui demande.

Enfin, faites vous plaisir. Vous faites partie d'une belle équipe, et vous travaillez tous sur le même projet. N'ayez pas peur de passer du temps devant votre écran à fignoler votre code. Ne vous contentez pas d'un simple « ça marche… ». Il faut faire mieux que bien. Car, de toute manière, le robot ne commencera à marcher en phase finale que dans le dernier mois avant la compétition. Prenez le temps de bien faire les choses. Il vaut mieux aller lentement et le faire qu'une seule fois, plutôt que se dépêcher et le refaire plusieurs fois.

N'oubliez pas que vous n'êtes pas seuls. Les membres de votre équipe sont aussi là pour vous aider, et inversement, même si ils n'occupent pas le même poste que vous. Soyez humbles, et apprenez à recevoir des conseils de la part des autres. C'est dans l'intérêt de tous.



ANNEXE 1 : FONCTION RAMENE CORRIGEE

Voici le code en entier du fichier BOU.C.

BOU.C

```
#include <io72314.h>
#include "bou.h"
#include <string.h>

/*
const int seuil_red =      10;
const int seuil_green =   10;
const int seuil_blue =     10;
const int seuil_box_x = 5;
const int seuil_box_y = 5;
 */
//const unsigned int dist_B1=30;
//const unsigned int angle_B1=45;
//const unsigned char sw_init_Klibre[] = {'s','w',' ','1',' ','1',' ','1',' ','1',0x0D};
//const unsigned char GM[] = {'g','m',0x0D};

//const unsigned char LUMIERE[]={'C','R',2,19,32};
const float angle= 0.4363324;   // angle en radian de la caméra par rapport à l'horizontal
const unsigned int H = 33;// hauteur de la caméra.
const unsigned char RS[]={'R','S',0x0D};
const unsigned char SW[]={'S','W',0x0D};
const unsigned char PM1[]={'P','M',' ','1',0x0D};
const unsigned char L12[]={'L','1',' ','2',0x0D};
const unsigned char L11[]={'L','1',' ','1',0x0D};
const unsigned char L10[]={'L','1',' ','0',0x0D};
const unsigned char TC1[]={'T','C',0x0D};
//const unsigned char TW[]={'T','W',0x0D};
const unsigned char TCred[] = {'t','c',' ','1','7','0',' ','2','4','0',' ','1','6',' ','1','6',' ','1','6',' ','1','6',0x0D};
const unsigned char alive[] = {0x0D};
const unsigned char MM[] = {'M','M',' ','0',0x0D};
const unsigned char MM1[] = {'M','M',' ','1',0x0D};
//const unsigned char TCb[] = {'T','C',' ','1','6',' ','1','7',' ','6','0',' ','1','2','0',' ','1','4','0',' ','2','4','0',0x0D};
//const unsigned char DIST[] = {'S','W',' ','1','0',' ','1',' ','7','0',' ','1','4','3',0x0D};
```

```
                                //4 5                70
const float tan_angle=0.932515; //tan[1/2angle ouverture caméra + angle cam/robot=]
const char depart_tortue = 0;
const char depart_position_x = 0;
const char depart_position_y = 180;


extern unsigned char tableau_mvt[21];
extern int compteur_mvt;
// 1-> rotation gauche
// 2-> rotation Droite
// 3-> avance
// 4-> recule
// 5-> stance
extern unsigned int temp1; //u
extern unsigned int temp2; //d
extern unsigned int temp3; //c
extern unsigned int boucle;
extern unsigned int sortie;
extern unsigned int ceinture;
extern unsigned char regt;
extern unsigned char regt1;
extern int para1;
extern int para2;
extern int para3;
extern int para4;
extern long paralong;
extern unsigned int mot[6];
extern unsigned int mot1[6];
extern unsigned int ref[6];


extern float a;
extern float b;
/*odometrie:*/  extern int tortue;
                //extern char position_x;
                extern char position_y;
```

```
/*
unsigned int opticien(unsigned char chaine)
{
	if ((chaine > 47) && (chaine < 58))
	{
		return(chaine - 48);
	}
	else
	{
		return(chaine);
	}
}
*/




// **************  fonction send ********************
void send(unsigned char commande[])
{
	boucle = 0;
	  while(commande[boucle]!=0x0D)
	  {
		do
			{ regt = SCISR;}
		while((regt&0xC0) == 0x00);    // acess to SCISR register
		SCIDR = commande[boucle];    // write byte to send to SCIDR; clearing of TDRE bit
		para1 = 170;
		tempo();
		boucle++;
	  }


		do
			{ regt = SCISR;}
		while((regt&0xC0) == 0x00); // acess to SCISR register
```

```
                SCIDR = 0x0D;
                para1 = Temp;           // write byte to send to SCIDR; clearing of TDRE bit
                tempo();


}



////////////////////////////////////////////////////////////////////////////////
//\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
////////////////////////////////////////////////////////////////////////////////

void receive()
{
        ceinture = 0;
        for(boucle = 0;boucle<5;boucle++)
        {
        do
                {regt = SCISR; }
        while((regt&0x20) == 0x00);
        para1 = Temp;
        tempo();
        }
        regt1 = SCIDR;
        para1 = 0;
        while ((regt1 != 0x0D)/* && (ceinture < 21)*/)
    {
                do
                {
                regt = SCISR;
                regt1 = SCIDR;
                para1++;
                }

                while(((regt&0x20) == 0x00) && (para1 < 32000)); /*&& (regt1 <0x30) && (regt1 > 0x39));*/
                tableau_mvt[ceinture] = regt1;
```

```
                ceinture++;
        }
//      tableau_mvt[ceinture] = '\0';
}


void ack_nck(void)
{
        regt1 = SCIDR;
        while ((regt1 != 0x0D) && (ceinture < 21))
    {
                do
                {
                regt = SCISR;
                regt1 = SCIDR;
                para4++;
                if (para4 == 32000)
                {_asm("JP suite");}
                }
                while((regt & 0x20) == 0x00); /*&& (regt1 <0x30) && (regt1 > 0x39));*/
                tableau_mvt[ceinture] = regt1;
                ceinture++;

        }
        _asm("suite:");
}


//  ************tempo : 170 correspond à peu près au temps qu'il faut pour envoyer un octet à 19200 b/s
void tempo()
{
        for(sortie = 0;sortie<para1;sortie++)
        {
                sortie = sortie;
        }
}
```

```
void initst7_seul(void)
{
	//PEDDR = 0x00; initialisé dans hctl.c
	//PEOR = 0x00;

	MISCR1 = 0x10;				//IS11|IS10|MCO|IS21|IS20|CP1|CP0|SMS| normal mode, fcpu=fosc/2 faut-il le faire???????
					//Initialisation et définition du protocole de transmission/reception du st7

	SCIBRR = 0x00;				//pour le st7 seul on utilise les registres etpr et erpr
	SCICR1 = 0x00;
	SCICR2 = 0x0C;				//00001100 TIE|TCIE|RIE|ILIE|TE|RE|RWU|SBK TE=1:transmitter enable  RE=1:receiver enable
	SCISR = (SCISR|0x20);		//initialise le fameux bit pour bien recevoir à 0


	SCIERPR = 0x07;//0x08;				//fcpu/(16*ETPR) pour le ST7 seul, on  utilise 0x08
	SCIETPR = 0x07;//0x08;

}

void initst7_autobaud(void)
{
	PEDDR = 0x00; //initialisé dans hctl.c
	PEOR = 0x00;

	MISCR1 = 0x10;				//IS11|IS10|MCO|IS21|IS20|CP1|CP0|SMS| normal mode, fcpu=fosc/2 faut-il le faire???????
					//Initialisation et définition du protocole de transmission/reception du st7

	SCIBRR = 0x00;				//pour le st7 seul on utilise les registres etpr et erpr
	SCICR1 = 0x00;
```

```
        SCICR2 = 0x0C;                          //00001100 TIE|TCIE|RIE|ILIE|TE|RE|RWU|SBK
TE=1:transmitter enable  RE=1:receiver enable
        SCISR = (SCISR|0x20);           //initialise le fameux bit pour bien recevoir à 0

      para2  = 0x01;
        while (para4 < 32000)
        {
                para4 = 0;
                SCIERPR = para2;                        //fcpu/(16*ETPR) pour le ST7 seul, on  utilise
0x08
                SCIETPR = para2;
                para2++;
                send(alive);
                para1 = Temp;
                tempo();
                send(alive);
                para1 = Temp;
                tempo();
                ack_nck();
                para1 = 10 * Temp;
                tempo();
        }

}

void initst7_plaq(void)
{
        //PEDDR = 0x00;
        //PEOR = 0x00;

        MISCR1 = 0x18;                          //IS11|IS10|MCO|IS21|IS20|CP1|CP0|SMS| normal mode,
fcpu=fosc/2 faut-il le faire???????
                                        //Initialisation et définition du protocole de transmission/reception
du st7

        SCIERPR = 0x00;
        SCIETPR = 0x00;
```

```
	SCIBRR = 0xC0;					//11000000;	// Baud Rate Generator: SCP1|SCP0|SCT2|SCT1|SCT0|SCR2|SCR1|SCR0
					// vitesse transmit = Fcpu/(32*PR*TR)   Fcpu=Fosc/2=16MHz/2  PR=13, TR=1 RR=1
					// 1100 0000 => 19 200 bauds
	SCICR1 = 0x00;
	SCICR2 = 0x0C;					//00001100 TIE|TCIE|RIE|ILIE|TE|RE|RWU|SBK TE=1:transmitter enable  RE=1:receiver enable
	SCISR = (SCISR|0x20);		//initialise le fameux bit pour bien recevoir à 0



}


void initcam(void)
{
	tortue = depart_tortue;
	position_y = depart_position_y;
	para1 =Temp;
	tempo();
	send(RS);	// reset caméra
	para1 = 60*Temp;
	tempo();
	send (alive);
	para1 = 60*Temp;
	tempo();
	send(SW);
	para1 = 60*Temp;
	tempo();
	send(alive);
	para1 = 60*Temp;
	tempo();
	send(PM1);	//poll mode on : 1 seule trame renvoyée.
	para1 = 60*Temp;
	tempo();
}
```

```
//******fonction qui sert juste à faire clignoter la diode rouge de la caméra
void alerte(void)                       //nombre de pulse désirés en entré
{
        for(temp1=0;temp1<ceinture;temp1++)
        {
                send(L11);
                para1 = 10000;
                tempo();
                send(L10);
                para1 = 10000;
                tempo();
        }

}


void addidas (void)
{
        boucle=0;
        while(((tableau_mvt[boucle] < 0x30) || (tableau_mvt[boucle]>0x39)) && (tableau_mvt[boucle] != 0x0D))

          {boucle++;}

      //on enleve les espaces
      ceinture = 0;

        do
        {
                mot[ceinture] = 0;
                do
                {
/*remplace opticien*/   if ((tableau_mvt[boucle] > 47) && (tableau_mvt[boucle] < 58))
/*remplace opticien*/   {tableau_mvt[boucle] = tableau_mvt[boucle] - 48;}
```

```
                        //mot[ceinture] = 10*mot[ceinture] + opticien(tableau_mvt[boucle]);
/*remplace opticien*/     mot[ceinture] = 10 * mot[ceinture] + tableau_mvt[boucle];

/*remplace opticien*/   if (tableau_mvt[boucle] < 10 )
/*remplace opticien*/   {tableau_mvt[boucle] = tableau_mvt[boucle] + 48;}
                        boucle++;



        }
        while((tableau_mvt[boucle] != 32) && (tableau_mvt[boucle] != 0x0D));
          do
          {boucle++;}
          while((tableau_mvt[boucle] == 32) && (tableau_mvt[boucle] != 0x0D));
        ceinture++;
        }
        while((ceinture <= para2) && (tableau_mvt[boucle] != 0x0D));
}

void addidas_bar (void)
{
        boucle = 0;
        for(ceinture = 0;ceinture<para2;ceinture++)
    {
        para3 = mot[ceinture];
        temp1 = para3%10;
        para3 /= 10;
        temp2 = para3%10;
        para3 /= 10;
        temp3 = para3%10;
                if (temp3>0)
                {
                        tableau_mvt[boucle] = temp3 + 0x30;
                        boucle++;
                        tableau_mvt[boucle] = temp2 + 0x30;
                        boucle++;
                        tableau_mvt[boucle] = temp1 + 0x30;
                        boucle++;
```

```
                tableau_mvt[boucle] = ' ';
        }
        else if (temp2>0)
        {
                tableau_mvt[boucle] = temp2 + 0x30;
                boucle++;
                tableau_mvt[boucle] = temp1 + 0x30;
                boucle++;
                tableau_mvt[boucle] = ' ';
        }
        else
        {
                tableau_mvt[boucle] = temp1 + 0x30;
                boucle++;
                tableau_mvt[boucle] = ' ';

        }
        boucle++;

    }
    tableau_mvt[boucle - 1] = 0x0D;

}



//***********fonction qui ramène les coordonnées de la balle la plus proche*****************//
void ramene()
{

    boucle=0;
    ceinture=0;
    send(alive);
    para1 = 20*Temp;
    tempo();
    send(SW);
    para1 = 100*Temp;
```

```
tempo();
send(alive);
para1 = 20*Temp;
tempo();
send(SW);
para1 = 100*Temp;
tempo();
send(alive);
para1 = 6*Temp;
tempo();

a = 0;
do
{
        send(TCred);
        para1 = Temp;
        tempo();
        receive();
        para1 = Temp;
        tempo();
        send(alive);
        para1 = 20*Temp;
        tempo();
        a ++;
}
while ((a < 7) && (tableau_mvt[2] == '0'));

if (tableau_mvt[2] == '0')
{
         _asm("JP pa_balle");
}
para2 = 6;
addidas();

//////////////////////////////////////////
//////tribal
```

```
    _asm("tatoo3:");

    if ((mot[5] - mot[3]) > 20)    //coupage en deux vertical
    {mot[5] = mot[1];
    trans2();
    para2 = 4;
    addidas_bar();
    boucle = 17;
    do
    {tableau_mvt[boucle + 3] = tableau_mvt[boucle];
            boucle--;}
    while (boucle > 0);
    tableau_mvt[3] = tableau_mvt[0];
    tableau_mvt[0] = 's';
    tableau_mvt[1] = 'w';
    tableau_mvt[2] = ' ';
    send(tableau_mvt);
para1 = 100;
tempo();
    send(alive);
    para1 = 20*Temp;
    tempo();
    send(tableau_mvt);
para1 = 100;
tempo();
    send(alive);
    para1 = 60*Temp;
    tempo();
    a = 0;
    do
    {
            send(TCred);
            para1 = Temp;
            tempo();
            receive();
            para1 = 10*Temp;
            tempo();
```

```
                send(alive);
                para1 = 20*Temp;
                tempo();a ++;
        }
        while ((a < 7) && (tableau_mvt[2] == '0'));
        if (tableau_mvt[2] == '0')
        {_asm("JP pa_balle");}
        else
        {para2 = 6;
        addidas();
        _asm("JP tatoo3");}
        }




        _asm("tatoo:");
        para4 = mot[4] - mot[2];                    // on calcule la variance dX.
        if (para4 > 15)                             // on fait la récurrence tant qu'on ne tombe pas sur 1 seule
balle.(dX<3)
        {       mot[4] = mot[2] + para4 / 2 + 1;    // on travaille sur la première moitié du cadre visioné
                trans();
                ////////////////////////////////////////////////////////////////////
                ///////cherche
                _asm("tatoo2:");
//              mot[5] = mot[3] + 6;
                trans2();
                para2 = 4;
                addidas_bar();
                boucle = 17;
                do
                {tableau_mvt[boucle + 3] = tableau_mvt[boucle];
                        boucle--;}
                while (boucle > 0);
                tableau_mvt[3] = tableau_mvt[0];
                tableau_mvt[0] = 's';
                tableau_mvt[1] = 'w';
```

```
		tableau_mvt[2] = ' ';
		send(tableau_mvt);
		para1 = 100*Temp;
		tempo();
		send(alive);
		para1 = 20*Temp;
		tempo();
		send(tableau_mvt);
		para1 = 100*Temp;
		tempo();
		send(alive);
		para1 = 10 * Temp;
		tempo();

		temp1 = 0;
		do
		{
			temp1++;
			send(TC1);
			para1 = Temp;
			tempo();
			receive();
			para1 = 10 * Temp;
			tempo();
		} while ((temp1< 7) && (tableau_mvt[2] == '0'));

		para2 = 6;
		addidas();
		if (mot[0] == 0)
			{ceinture = 0;}
		else
			{ceinture = 1;}
	////////fin cherche
	//////////////////////////////////////////

		if (ceinture == 1)		//on lui demande si elle voit la balle la plus proche.
			{_asm("JP tatoo");
```

```
                }
                else
                {
                        ref[2] = ref[4];
                        ref[4] += para4/2;          // on travaille sur la seconde moitié du cadre visionné
                        mot[2]= ref[2];
                        mot[3]= ref[3];
                        mot[4]= ref[4];
                        mot[5]= ref[5];
                        if (ref[4] < 81)
                        {_asm("JP tatoo2");}

                }
        }
// on définit les nouvelles dimensions du cadre et on relance la fonction jusqu'à voir une seule balle.
   /*   send(MM1);
        para1=6*Temp;
        tempo();
        send(TC);
        para1=temp;
        tempo();
        receive();
        para1=temp;
        tempo();
        send(MM);
        para1=6*Temp;
        tempo();
        para2=2;
        addidas();
        */ //code de remplacement si jamais on manque de place RAM




/////fin tribal


      if (tableau_mvt[2] != '0')
      {/*présence de balle*/
```

```
        _asm("jp qqpart");}



    //////////////////////
//      CODE D'EVITEMENT
        //////////////////////

    _asm("pa_balle:");
        if (b == 0)
        {
                tableau_mvt[0] = '1';
                tableau_mvt[1] = 30;
                tableau_mvt[2] = '0';
                b++;
        }
        else
        {
                if (b < 7)
                {
                        tableau_mvt[0] = '2';
                        tableau_mvt[1] = 60;
                        tableau_mvt[2] = '0';
                        b++;
                }
                else
                {
                        tableau_mvt[0] = '3';
                        tableau_mvt[1] = 30;
                        tableau_mvt[2] = '0';
                        b = 0;
                }
        }
        compteur_mvt = 0;
        while (tableau_mvt[compteur_mvt] !='0')
        {
                mouvement2();
```

```
                compteur_mvt = compteur_mvt +2;
        }

        _asm("jp do_it");

    ////////////////////////
//      FIN DE CODE D'EVITEMENT
        ////////////////////////

    _asm("qqpart:");
        a = mot[0];
        a -= 51;
        b = mot[1];
        b += H/tan_angle;// longueur (en pixel?) manquant à la latitude.
        b = a/b;
        a = angle;
        cos();
        a = b / a;//a=tan(theta)/cos(angle);
        arctan();
        if (a > 0)
                {tableau_mvt[0] = '2';
                a = a ; // à cause du décalage de la vision de la caméra...
                }
        else
        {
                tableau_mvt[0] = '1';
                a = -a;
        }
        b = a * 180 / (3.141592654); // (1) a en radian, transformation en degres dans b pour gagné le temp de calcul en reconcersion en radians
        para3 = b;
        tableau_mvt[1] = para3;
        b = mot[1];
        compteur_mvt= 2;
        stance();
        _asm("do_it:");
        compteur_mvt=4;
```

```
}



void stance(void)
{
        send(L11);
        para1 = 7*Temp;
            tempo();
            send(alive);
            tableau_mvt[compteur_mvt] = '3';
            cos();
            a = (43 + 0.3 * b + 0.002 * b * b) / a;  // 25.7 0.1898 0.00296
            para3 = a;
            tableau_mvt[compteur_mvt+1] = para3;
            tableau_mvt[compteur_mvt+2] = '0';
            send(L10);
            b = 0;

//test qualif 3
            compteur_mvt = 0;
        regt1 = '0';
        PFDR = 0x10;
        while (tableau_mvt[compteur_mvt] !='0')
            {
            mouvement2();
            compteur_mvt = compteur_mvt +2;
            if (regt1 == '1')
            {_asm("JP pa_tire");}
            }
        PFDR = 0x00;
            fire();
            compteur_mvt = 0;
            while (tableau_mvt[compteur_mvt] !='0')
            {
            mouvement2();
```

```
        compteur_mvt = compteur_mvt +2;
        if (regt1 == '1')
        {_asm("JP pa_tire");}

        }
        _asm("pa_tire:");

//fin test qualif 3

}
```

ANNEXE 2 : FICHIER MATH.C

Voici le fichier contenant toutes les fonctions trigonométriques que nous avons utilisées durant l'élaboration de notre code.

```
#include "bou.h"


#define NB_PULSE_ROT 525              // rapport de rotation = nb-pulse_demi-tour/PI
#define RAPPORT_MAX_ARCTAN 0.912    // Solve(x-x^3/3+x^5/5=PI/4,x) dl de arctan à l'ordre 5
#define CENTRE 50 //définit où se trouve l'axe du robot à l'image

extern float a;
extern int para1;


//nombre de pas pour aller jusqu'au pixel j dans l'axe du robot
//dj[j]= hauteurcam*tan(theta1+j(theta2-theta1)/143)
// avec tan(theta1) = distance_min_vue/hauteurcam   et tan(theta2) = distance_max_vue/hauteurcam
//const int
dj[143]={660,665,671,676,682,688,693,699,705,711,717,723,729,735,741,747,753,759,766,772,779,785,792,798
,805,812,819,826,833,840,847,854,862,869,877,884,892,899,907,915,923,931,939,948,956,965,973,982,991,100
0,1009,1018,1027,1036,1046,1056,1065,1075,1085,1095,1106,1116,1127,1138,1148,1160,1171,1182,1194,1205,
1217,1229,1242,1254,1267,1280,1293,1306,1320,1333,1347,1362,1376,1391,1406,1421,1437,1452,1468,1485,1
502,1519,1536,1554,1572,1590,1609,1628,1647,1667,1688,1709,1730,1752,1774,1796,1820,1843,1868,1893,19
18,1944,1971,1998,2027,2055,2085,2115,2147,2179,2212,2246,2280,2316,2353,2391,2431,2471,2513,2556,260
0,2646,2693,2742,2793,2846,2900,2957,3016,3077,3140,3206,3275};

//largeur en pas d'un pixel suivant la ligne j formaté en entier : contenu du tableau à diviser par 16
// suivant une loi inverse lpixel[j]=cst/(aj+b)
//const float lpixel[143]={}
//const int
lpixel[143]={135,136,136,137,138,139,139,140,141,142,143,143,144,145,146,147,148,149,150,150,151,152,153,
154,155,156,157,158,159,160,161,162,163,165,166,167,168,169,170,171,173,174,175,176,178,179,180,182,183,
184,186,187,188,190,191,193,194,196,198,199,201,202,204,206,208,209,211,213,215,217,219,221,223,225,227,
229,231,233,235,238,240,242,245,247,250,252,255,257,260,263,266,269,272,275,278,281,284,287,291,294,298,
301,305,309,313,317,321,325,330,334,339,343,348,353,358,364,369,375,380,386,392,399,405,412,419,426,434,
441,450,458,467,475,485,495,505,515,526,538,549,562,575,589,603};
/*
```

```
unsigned char min(unsigned char i, unsigned char j)
{if (i>j) return(j); else return(i);}
/*
unsigned char abs(signed char v)
{
if(v<0) return(-v);
else return(v);
} */

/*
float absf(float v)
{
if(v<0) return(-v);
else return(v);
}
*/

void cos (void)
{
//angle en radian
a = a*a/2;

a = 1 - a/2  +a*a/24 - a/720*a*a;
}
void sin ( void)
{
//angle en radian
a = a - a*a*a / 6; //+ a/ 120*a*a*a*a; plante acces memoire interdite
}


void arctan(void)  // le rapport est supposé inférieur à 1
{
//DL ordre 5 => erreur de 1% avec un rapport de 0.6 ; de 5,9% à la limite RAPPORT_MAX_ARCTAN=0.912
para1 = 0;
if (a < 0)                  ///
        {a = - a;           ///
```

```
        para1 = 0x02;} ///

//if (absf(a) >1)
if (a > 1)                  ///
{
   a = 1/a;
   para1 ++;
}
//if ( absf(a) > RAPPORT_MAX_ARCTAN)
if (a > RAPPORT_MAX_ARCTAN)
        a = PI/4;
else
{
        if (para1 >= 2)  ///
        { a = -a;          ///
        para1 -= 2;}              ///

        if (para1 == 0)
                a = (a - a*a*a/3 + a*a*a*a*a/5);
        else
                a = (PI/2 - a + a*a*a/3 - a*a*a*a*a/5);
}
}


void tan(void)
{
        a = a + a*a*a/3 + a*a*a*a*a/30 + a*a*a*a*a*a*a/120;
```

ANNEXE 3 : CAMERA

Description des Functions

Les paramètres de la liaison série sont les suivants :
115200 baud
8 bits de données
1 bit de stop
Pas de parité
Pas de contrôle de flux (pas de protocole Xon/Xoff ou Hardware)
Toutes les commandes sont envoyées sous forme de caractères ASCII visualisables(123 donne « 123 »). Après une succession de commandes correctement transmises, la chaîne « ACK » est retournée. Si un problème apparaît dans la syntaxe de la commande envoyée, ou si une erreur de transmission est apparue, une chaîne « NCK » est retournée. Après chaque ACK ou NCK, un « \r » est retourné. Lorsqu'une commande vide (« \r » suivi de « : ») est retournée, la caméra est alors dans un état d'attente d'une autre commande. Les espaces blancs comptent de manière significative, ils sont utilisés pour séparer les arguments passés en paramètre des commandes. Le « \r » (code ASCII 13= retour chariot) est utilisé pour valider une commande et terminer l'action de la commande envoyée.
Si la trame de données transmises est trop rapide, il est possible d'utiliser différents taux de transferts (voir les paramètres de la liaison série).

Description des commandes :
\r
Cette commande est utilisée pour régler le module caméra en mode inactif. Comme toutes les autres commandes, la chaîne de comfirmation « ACK » ou d'erreur « NCK » est renvoyée si un problème est apparu.
Après confirmation du mode inactif, le module caméra attend d'autres commandes, ce qui est visible par le caractère "« « affiché dans l'hyper terminal . Lorsque le module est dans un état inactif, un « \r » seul retournera la chaîne « ACK » suivi de « \r » et un affichage de « : ».
Exemple de vérification du bon fonctionnement de la caméra dans l'état inactif
:
ACK
:

CR [reg1 valeur1 [reg2 valeur2 … reg16 valeur16]] \r
Cette commande règle les registres internes de la caméra (Camera Register) directement. Les emplacements mémoire de stockage des registres et les valeurs possibles de réglage sont accessibles dans la documentation Omnivision. Toutes les données envoyées par cette

commande peuvent être sous forme décimale à condition que la caméra ait été préalablement configurée en « raw mode ». Il est possible d'envoyer plus de 16 valeurs par registre. Les valeurs initiales des registres ne sont pas réinitialisés entre chaque commande CR ; cependant, on peut modifier la configuration par défaut. Exécuter cette commande sans argument réinitialise la caméra, et remet tous les registres à leur valeur initiale.

Principaux réglages possibles :

Registre	Valeur
5 contraste	0-255
6 Eclairage (luminosité)	0-255
18 Mode de couleur	36 YCM* Automatique, réglage de blanc activé 32 YCM* Automatique, réglage de blanc désactivé. 44 RGB Automatique, réglage de blanc activé 40 RGB Automatique, réglage de blanc désactivé (valeur par défaut)
17 Vitesse d'horloge	2 17 fps (= traitement par seconde) -> valeur par défaut 3 13 fps 4 11 fps 5 9 fps 6 8 fps 7 7 fps 8 6 fps 9 5 fps 10 4 fps
19 Exposition automatique	32 Gain automatique activé que désactivé

* La composante rouge devient Cr avec approximation r-g. La composante verte devient Y avec une intensité approximative, la composante bleue devient Cb avec les approximations b-g.

RGB -> CrYCb
Y= 0.59G+0.31R+0.11B
Cr=R-Y
Cb=B-Y

DF \r
Cette commande renvoie une image en sortie du port série. C'est la seule commande qui, par défaut, peut envoyer des trames en caractères ASCII non interprétables à l'écran. Cette commande envoie une trame de type « F » qui est constituée des données vidéo colonne par colonne avec un octet de synchronisation d'image et un octet de synchronisation pour la colonne envoyée (cette donnée peut être lue et affichée par l'application java CMUcamGUI). Si le taux de données excède de beaucoup la vitesse maximum de traitement du

microprocesseur branché en sortie du module, l'image peut être envoyée colonne par colonne à la fois. Cela permet de n'observer que les changements lents sur ce que voit la caméra.
Pour obtenir une bonne image, il fa ut doubler les colonnes de pixels.
La caméra est capable d'envoyer une image de résolution maximale à vitesse maximale (17 colonnes par seconde) seulement, à 115200 bauds. Pour des taux de transferts plus longs, ou à 115200 bauds avec des temps d'attente entre chaque colonne, le taux d'envoi doit être diminué pour avoir une bonne résolution d'image.
Avec un gain automatique, à de plus faibles taux, l'image peut apparaître plus claire qu'habituellement.
Ceci est dû au fait que la caméra capture l'image plus lentement que la normale et prend donc plus de temps pour régler l'intensité de l'image. Essayez dans ce cas de régler manuellement le contraste et l'intensité.

Format d'une trame de données de type F :
1-nouvelle image
2-nouvelle colonne
3-fin de l'image
RGB (CrYCb), valeurs entre 16 et 240
1 2 rgb rgb … rgb rgb 2 rgb rgb … rgb rgb 3

Exemple de transfert d'une image envoyée à l'hyper terminal (ATTENTION : des caractères qui ne sont pas dans le code ASCII peuvent ne pas apparaître sous l'hyper terminal !) :
:DF
ACK
MaKP(U ….

DM valeur \r
Cette commande règle la temporisation (Delay) entre chaque trame envoyée sur le port série. La valeur doit être comprise entre 0 et 255. Une valeur de 0 (valeur par défaut) ne donne aucune temporisation et 255 règle la temporisation au maximum. Chaque unitée de valeur de temporistion correspond à peu près au temps de transfert d'un bit sur la liaison série, avec le taux de transmission choisi.

GM \r
Cette commande récupère la couleur dominante de l'image courante. La valeur renvoyée est comprise entre 16 et 240. Cette commande retourne donc une mesure de la plus grande déviation absolue de la couleur trouvée dans l'image.
La moyenne associée à la déviation peut être utile pour détecter un changement dans une image.
En mode YcrCb, le RGB est modifié en YcrCb.

Format d'une trame de type « S » (statistique) :
S Moyenne_Rouge Moyenne_Vert Moyenne_Bleue Deviation_Rouge Deviation_Vert Deviation_Bleue
Exemple :
Pour saisir la couleur moyenne de toute la fenêtre :
: SW 1 1 40 143
ACK
:GM
ACK

S 89 90 67 5 63
S 89 91 67 5 6 3

GV \r
Cette commande récupère la version courante du logiciel de la caméra (Get current Version). Elle retourne un ACK suivi de la chaîne de caractère donnant la version du logiciel.
Exemple :
:GV
ACK
CMUcam v1.12

HM actif\r
Cette commande configure la caméra en mode résolution semi-horizontale (Half_horizontal resolution Mode) pour la commande DF et la commande LM, à la récupération d'une image bitmap. Une valeur de 1 dans actif provoque seulement la récupération des colonnes de trames impaires de l'image. Une valeur de 0 dans actif (valeur par défaut) désactive ce mode.

I1\r
Cette commande permet d'utiliser le port de commande du servomoteur en entrée.
L'activation de cette commande retourne 1 ou 0 en fonction du niveau de tension appliqué sur le port du sevomoteur. Le port est forcé à 1 ; c'est pour cela qu'il est conseillé de le forcer à 0 ou de le laisser flottant pour changer son état. Le port du servomoteur peut alors être utilisé en sortie digitale (voir la commande S1).
Exemple :
:I1
ACK
1

L1 valeur\r
Cette commande est utilisée pour contrôler la LED de suivi. Elle accepte 0,1 ou 2 (valeur par défaut) en entrée. Une valeur de 2 active la LED de suivi en mode automatique. Dans ce mode, est durant le suivi, la LED s'allume lorsqu'elle détecte la présence d'un objet qui correspond à l'image à chercher, approximativement. Cette commande est utile pour le débuguage.
Exemple :
:L1 2
ACK
:L1 0
ACK

LM actif\r
Cette commande active le Mode Ligne (Line Mode) qui utilise le temps entre chaque image pour transmettre des informations plus détaillées correspondant à la description de celle-ci. Elle ajoute une trame de type C,M ou S. Ce mode est utilisé par les utilisateurs qui veulent des images plus complexes avec des données moins réduites. Tant que le taux d'image n'est pas compromis, le système actuel de traitement de données connecté en sortie du module caméra doit être suffisamment rapide, pour récupérer les données avec un système transfert rapide, ce qui n'est pas forcément évident avec les microcontrôleurs lents.
Effet du mode ligne sur les commandes TC et TW : lorsque le mode ligne est actif, à l'appel de la commande TC ou TW, ce mode renvoie une image bitmap en binaire. Le code du

bitmap commence avec la valeur 0XAA. Cette valeur ne peut donc pas apparaître dans la chaîne de données.
Cette chaîne est suivie d'une suite d'octets qui contient la valeur binaire de 8 pixels récupérés du côté haut-gauche jusqu'au côté bas-droite. La résolution verticale limitée par le temps de transfert des données horizontales, donc des lignes, pourraient être perdues lors de la récupération des données. En mode haute résolution, la résolution de l'image est de 80x48. L'image bitmap se termine par deux octets ayant chacun pour valeur 0xAA. Ceci est ensuite suivi de la trame de données standard de type C ou M (effectuée à une réolution plus faible).

Exemple d'exécution de la commande TC avec le mode ligne activé :
:LM1
:TC
(trame de données :AA XX XX XXX … XX XX AA AA) C 45 72 65 106 18 51
(trame de données :AA XX XX XXX … XX XX AA AA) C 46 72 65 106 18 52

Effet du mode ligne sur la commande GM :
Lorsque le mode ligne est actif, et que la commande GM est envoyée, le mode ligne envoie une trame de données (non facilement lisible) correspondant aux valeurs moyennes de chaque ligne traitée. Ces trames commencent par la valeur 0xFE et se terminent par 0xFD. Chaque octet de données comprise entre ces deux valeurs représente la valeur de la couleur dominante de chaque ligne traitée. De la même façon qu'avec le qu'avec le mode bitmap, la résolution verticale est divisée par 2, à cause du temps de transfert de la liaison série. A 17 image par seconde, et 115200 baud, toute autre ligne est retirée. A un taux moins élev é (que 115200 bauds)aucune ligne ne sera retirée.
Exemple de l'utilisation de la commande GM avec le mode ligne retiré :
:LM1
:GM
(trame de données ligne : FE XX XX … XX XX FD) M 45 56 34 10 15 8
(trame de données ligne : FE XX XX … XX XX FD) M 45 56 34 10 15 8

MM mode\r
Cette commande contrôle la masse moyenne (Middle Mass) qui additionne les coordonnées polaires à la donnée à rechercher. Une valeur de 0 dans le paramètre mode retire la masse moyenne, une valeur de 2 retire ce mode et active la sortie pwm du servomoteur qui essaie de centrer la caméra sur l'image à rechercher (voir la description du mode démo). Ce mode est utilisable dans le cas où vous voulez localiser quelques points d'un objet ou si l'image est trop bruitée pour avoir une fenêtre limitée. Pour commuter la direction du servomoteur, il est nécessaire de mettre le 2eme bit du paramètre mode à 1. Si le 3eme bit est à 1 la recherche de couleur retourne une trame de type « N », identique à une trame de type « M », mais qui contient uniquement la position courante du servomoteur, en première valeur retournée (voir la description des trames de données).
Exemple de la désactivation du mode MM :
:MM 0
ACK
:TC
ACK
C 38 82 53 128 35 98
C 38 82 53 128 35 98
C 38 82 53 128 35 98

NF actif\r
Cette commande permet de contrôler le filtre anti-bruit. Elle accepte les valeurs booléennes 1 (appliquée par défaut) ou 0. LA valeur 1 active le mode et la valeur 0 le désactive. Quand le filtre est actif, la caméra est plus stricte sur la sélection des pixels.
Exemple de désactivation du filtre anti-bruit :
:NF 0
ACK
:

PM mode\r
Cette commande active le module en mode statistique (Poll Mode). Si le paramètre mode = 1 , le mode statistique est activé. Si le paramètre mode = 0 , le mode statistique est désactivé. Au fonctionnement du mode statistique, seule une trame est retournée lorsqu'une fonction de traitement d'image est appelée. Cela peut être utile si vous voulez changer les paramètres en cours, ou si vous avez un processeur lent qui ne peut pas récupérer une image donnée au temps de transmission donné.

Exemple de récupération d 'une trame durant un temps donné :
:PM 1
ACK
:TC 50 20 90 130 70 255
ACK
C 38 82 53 128 35 98
:

RM bit_de_flag\r
Cette commande permet d'utiliser le mode de transfert série des trames ligne (Raw Mode) . Elle lit les valeurs des trois premiers bits (bits de poids faible) du paramètre pour la configuration de ce mode. Tous les bits réinitialisés à 0 mettent le mode ASCII par défaut, en visible. Si le bit 0 est à 1, toute donnée issue de la caméra est envoyée sous forme d'une suite d'octets les uns après les autres. Le format de cette trame de données sera modifiée pour ne pas inclure d'espaces ou formatés sous forme de texte ASCII lisible. Vous recevrez alors un octet égal à 255 au début de chaque trame, suivi de la trame de données, puis en dernier lieu, de la trame identifiant le type de données (i.e C pour une trame Couleur). Il n'y a pas de \r d'envoyé après chaque trame, donc vous devrez vous synchroniser avec l'octet de début (=255). Chaque octet égal à 255 présent dans la trame de données est automatiquement converti à 254, ceci afin d'éviter toute confusion avec l'octet de départ. Si le bit 1 est mis à A, la « ACK\r » et le « NCK \r » de configuration ne sont pas envoyés. Si le bit 3 est mis à 1, l'entrée sera lue comme une valeur correspondant à un octet de ligne. Dans ce mode, après que les valeurs des octets des deux commandes sont envoyés, envoyer un octet donnant le nombre d'arguments à suivre.(i.e. la commande DF suivie d'un octet =0pour pas d'argument). Aucun « \r » n'est nécessaire pour cette commande.
Si le bit 0 est à 1, les données en sortie de la caméra sont sous forme d'octets ligne.
Si le bit 1 est à 1, les « ACK\r » et les « NCK\r » sont retirés.
Si le bit 2 est à 1, l'entrée de la caméra est sous forme d'octet ligne.

Exemple d'une nouvelle trame qui fait une reconnaissance de valeur avec le mode ligne uniquement, en sortie (ATTENTION : cela peut ne pas être correctement interprété avec le terminal Windows par l'envoi de caractères ASCII non « visible s»):
:RM 1

ACK
:TC 50 20 90 130 70 255
ACKC>%k(aiCk$&,.L

RS \r
Cette commande réinitialise le module (ReSet)
Remarque : Après le reset, le 1er caractère est \r
Exemple :
:rs
ACK

CMUcam v1.12
:

S1 position\r
Cette commande vous permet de régler la position du servomoteur 1 5Set). Une valeur égale à 0 désactive le servomoteur et applique un signal bas sur le port d'E/S. 1à 127 positionne le servomoteur. Toute valeur égale ou supérieure à 128 active le port à l'éta haut. Pour que la commande servomoteur fonctionne correctement, la caméra doit être dans une boucle de reconnaissance ou dans une recherche de couleur dominante. Les valeurs 0 et 128 sont utilisées lorsqu'on veut utiliser le port « Servo » comme une sortie numérique (voir la commande I1). La commande « MM » peut activer ou désactiver le suivi automatique par servomoteur.

SM valeur \r
Cette commande est utilisée pour activer le mode « permutation » (Switching Mode) de la reconnaissance de couleur. Lorsque cette commande reçoit 0, c'est le mode par défaut qui est activé où une fonction de reconnaissance de couleur retourne une trame de type « C » ou « M ». Si la valeur envoyée est à « 1 « , la commande de recherche renverra l'image en alternant les trames de couleur et les trames de statistique de type « S ». Chaque trame de statistique est récupérée sur une partie unique de l'image, par rapport à la taille de l'aire, retournée de la commande de suivi. Si aucun objet n'est rattaché, aucune trame statistique (trame de type « S ») n'est renvoyée. Cela peut être idéal pour effectuer de la reconnaissance adaptative ou pour tout type de reconnaissance où vous voudriez retrouver les dimensions d'un objet à détecter.
Après que la 1er trame de recherche ait été retournée, la fenêtre reprend ses dimensions « normale » (complètes) pour les future trames .
Remarque : vous pouvez avoir seulement la moitié de la trame de couleurs par intervalle de temps.
Exemple de recherche des couleurs avec SM :

:SM1
ACK
:TC 200 255 0 30 0 30
ACK
C 2 10 12 60 10 70
S 225 20 16 2 3 1
C 5 60 20 30 12 100
S 225 19 17 1 2 1

C 0 0 0 0 0 0
C 0 0 0 0 0 0
C 0 0 0 0 0 0
C 5 60 20 30 12 100
S 225 19 17 1 2 1

SW [x y x2 y2]\r
Cette commande configure la taille de la fenêtre de l'image saisie (Sets the Window). Elle utilise les coordonnées cartésiennes x et y du coin supérieur gauche suivi des coordonnées du coin inférieur droite de la fenêtre comprise dans l'image. L'origine est localisée à partir du coin supérieur gauche de la vue entière de la caméra. Cette commande peut être sollicitée avant d'envoyer une commande de traitement d'image pour limiter
Le champ de vision. Si aucun argument n'est passé, la commande retourne la fenêtre complète par défaut, soit : 1,1,80,143.
Exemple pour ne prendre en compte qu'une demi-image issue de la caméra :
SW 35 65 45 75
ACK
:

TC [Rmin Rmax Gmin Gmax Bmin Bmax]\r
Cette commande commence par rechercher une couleur (Track a Color).Elle suit dans les limites de valeurs de couleurs RGB (ou CrYCb) données, et renvoie une trame de type « M » ou « C »(voir la commande MM). La trame de type C la plus petite comprend une boite limite qui contient les pixels de la couleur définie comme étant à rechercher, le nombre de pixels trouvés (échelle : la valeur actuelle est (pixels+4)/8) qui ressemblent aux valeurs données comprises dans les limites, et un taux de confiance. La trame renvoyée qui est, par défaut, de type « M », contient aussi l'information sur le centre de gravité (masse ?) de l'objet . La résolution de l'image traitée est de 80x143. Les valeurs en x sont comprises entre 1 et 80 et les valeurs en y sont comprises entre 1 et 143. Une trame uniquement constituée de 0 n'indique aucune couleur comprise dans ces limites n'est détectée. La valeur de confiance est le taux entre le nombre de pixels qui est contenu dans les limites données et l'aire de la boite couleurs limites. Cette valeur est comprise entre 0 et 255. Généralement, une valeur supérieure à 50 peut être considérée comme une valeur « de confiance » pour un objet simple. Une valeur de 8 ou moins peut être considéré comme insuffisant. Appelée sans argument, cette commande est appliquée avec les paramètres de précédents.
<u>Trame de type « M » :</u>
M mx my x1 y1 x2 y2 pixels confiance\r

<u>Trame de type « C » :</u>
C x1 y1 x2y2 pixels confiance\r

Exemple de recherche/suivi de couleurs avec les paramètres par défaut :
:TC 130 255 0 0 30 30
ACK
M 50 80 38 82 53 128 35 98
M 52 81 38 82 53 128 35 98
M 51 80 38 82 53 128 35 98

TW\r

Cette commande recherche les couleurs au centre de l'image (Track Window). Après que la couleur dans la fenêtre est récupérée, la fonction de suivi de couleur est appelée avec ses paramètres et appliquée à toute l'image. Cela peut être pratique pour observer et rechercher la présemce d'un objet devant la caméra. Comme elle appelle la recherche de la couleur actuelle, elle retourne le même type de trame : « C » ou « M ».
Remarque : votre réglage de fenêtre sera seulement utilisée pour récupérer la couleur à rechercher et ensuite la fenêtre retournera à une résolution de 80x143.
Exemple :
:TW
ACK
S 240 50 40 12 7 8
C 2 40 12 60 10 70
C 3 41 12 61 11 70
C 2 40 12 60 13 70
C 3 42 12 62 9 70
C 4 45 12 60 8 70

DESCRIPTION DES TRAMES DE DONNEES EN SORTIE

Toutes les trames de donnée en sortie sont au format ASCII visible mis à part les trames d'image de type « F » (Frame) et les trames de préfixe.
ACK : c'est la chaîne standard indiquant la bonne transmission des données.
NCK : c'est la chaîne standard indiquant un problème de transmission des données.

Trame de type « C » :
C'est la trame de retour lors de l'appel d'un commande de recherche/suivi.

x2 : valeur de la coordonnée en x du coin supérieur gauche
y2 : valeur de la coordonnée en y du coin supérieur gauche
x1 : valeur de la coordonnée en x du coin inférieur droite
y1 : valeur de la coordonnée en y du coin inférieur droite
pixel : nombre de pixels de la région recherchée, délimitée et décomposée en 255(pixel+4)/8
confiance : (nombre de pixels /surface de l'aire considérée)*256 du rectangle délimité , et décomposé en 255.

C x1 y1 x2 y2 pixels confiance\r

Trame de type « F » :
1-nouvelle image
2-nouvelle colonne
3-fin de l'image
RGB (CrYCb) valeurs entre 16 et 240.

RGB (CrYCb) représente les valeurs de la couleur de chaque pixel. Chaque pixel est compris entre le rouge et le bleu.
176 colonnes de trame RGB (CrYCb) (représente 352 pixels).
144 lignes

Pour affiche l'image avec un aspect correct, doublez chaque colonne, ainsi la résolution de l'image finale est de 352x144. la trame de données ne doit pas commencer avec un « F » et envoie uniquement des données ligne.

1 2 rgb rgb …. rgb rgb 2 rgb rgb … rgb rgb 3

Trame de type « M » :
C'est une trame de retour d'une commande de couleur avec le mode « Masse Moyenne » (Middle Mass) activé.

mx : la valeur en x de la masse moyenne
my : la valeur en y de la masse moyennne
x1 : valeur en x du coin supérieur gauche
y1 : valeur en y du coin supérieur gauche
x1 : valeur en x du coin inférieur droite
y1 : valeur en y du coin inférieur droite
pixel : nombre de pixels de la région recherchée, délimitée et décomposée en 255(pixel+4)/8
confiance : (nombre de pixels /surface de l'aire considérée)*256 du rectangle délimité , et décomposé en 255.

M mx my x1 y1 x2 y2 pixels confiance\r

Trame de type « N »
C'est la même trame que celle de type « M » avec en plus la position du servomoteur.

N spos mx my x1 y1 x2 y2 pixels confiance\r

Trame de type « S » :
Il s'agit d'une trame statistique qui donne des informations sur la vue de la caméra.

Rmean- valeur dominante du rouge (ou Cr) de la fenêtre courante
Gmean- valeur dominante du vert (ou Y) de la fenêtre courante
Bmean- valeur dominante du bleu (ou Cb) de la fenêtre courante
Rdeviation- déviation de rouge ou Cr trouvé dans la fenêtre courante.
Gdeviation- déviation de vert ou Y trouvé dans la fenêtre courante.
Bdeviation- déviation de bleu ou Cb trouvé dans la fenêtre courante.

S Rmean Gmean Bmean Rdeviation Gdeviation Bdeviation\r

*deviation : Moyenne de la différence absolue entrelespixels et la moyenne de la région.

Trame d'image eu format bitmap préfixé du mode ligne (Line Mode)
Cette trame est uniquement une ligne s'octets. Elle commence avec la valeur hexadécimale 0xAA, puis elle se poursuit avec une chaîne d'octets, dans laquelle chaque octet contient un masque/boite de 8 pixels commençant du côté haut-gauche est se terminant au côté bas-droite (chaque bit présent dans chaque octet représente un pixel). La bitmap se termine alors avec 0xAA. 0xAA n'est jamais présent dans les données, car cela signifierait la fin de transmission de l'image bitmap. Lorsque la transmission est terminée, une trame de recherche habituelle peut suivre.

(trame de données :AA XX XX XXX … XX XX AA AA) C 45 72 65 106 18 51
(trame de données :AA XX XX XXX … XX XX AA AA) C 46 72 65 106 18 52

Récupération de la trame dominante suivi du mode ligne (Line Mode).
Cette trame renvoie la couleur dominant de chaque ligne qui a été traité. Ces trames commencent avec 0xFE et se terminent avec 0xFD. Chaque octet de données compris entre ces valeurs représente la valeur de la couleur dominante, pour chaque ligne.
A cause du temps de transfert, la résolution maximale est divisée par deux. Après que toutes les lignes ont été traitées, une trame normale peut suivre.

ANNEXE 5 : @ UTILES

dresses utiles :
Pour plus de renseignements sur les possibilités et options de cette caméra, voir le site :
http://www.lextronic.fr/cmucam/PP.htm

Pour d'autres modules utiles, communications, boussole électronique… :
http://www.lextronic.fr

Achat des composants :➔ *Livraison le lendemain*.
http://international3.farnell.com/FR/Welcome/welcome-frame-unregistered.jhtml

source : rapport technique caméra 2002/2003.